电脑学习机上机指导系列丛书之六

汇编语言

于　春　张新莲　李悦国　编著

電子工業出版社
Publishing House of Electronics Industry

内 容 提 要

本书是电脑学习机上机指导系列丛书(共十册)的第六册。重点介绍了6527的寻址方式、6527的指令系统、汇编与BASIC、简单程序和分支程序设计、循环程序设计、子程序设计、堆栈程序设计、系统资源的利用等。通过有效地使用,可编写高效的应用程序,提高编程水平。

书　　名:汇编语言
著　　者:千春　张新莲　李悦国
责任编辑:施玉新
特约编辑:董　玲
印 刷 者:北京市顺义李史山胶印厂印刷
出版发行:电子工业出版社出版、发行　URL:http://www.phei.co.cn
北京市海淀区万寿路173信箱　邮编100036　发行部电话:68214070
经　　销:各地新华书店经销
开　　本:787×1092 1/32　印张:8.875　字数:206千字
版　　次:1997年5月第1版　1997年5月第1次印刷
印　　数:5000册
书　　号:ISBN 7-5053-3530-8/TP·1426
定　　价:12元

凡购买电子工业出版社的图书,如有缺页、倒页、脱页者,本社发行部负责调换
版权所有·翻印必究

前　言

家庭电视游戏业的发展，突飞猛进、一日千里。随着游戏机的普及，各种以任天堂系列 8 位游戏机为主机、配备键盘而构成的“普及型家庭电脑学习机”也迅速推出。目前生产厂已有三十余家，产品品种近百个。较有名的品牌有：中山小霸王、兰州金字塔、北京裕兴以及深、广一带的金钥匙、和声、科达、科特、美猴王、宏源、智源等，产品档次也已拉开了低、中、高三个级别。有关资料表明，目前我国电脑学习机的拥有量已逾 800 万台。

现在，学习机的功能日趋完善，实用性越来越强。除小霸王还继续维持其寓教于乐的游戏特点以外，各大生产厂家都向提高学习机的实用性进军。其工作系统已由从日本引进时的单一的 F BASIC（整数）语言发展到中文 BASIC（浮点）语言、LOGO 语言，汇编语言、中文编辑、文字处理等系统。就硬件来说，94 年初裕兴率先在其 YX4 型机上推出标准微机 101 键盘，随后和声、科达、金钥匙等立即推出了仿 101 键盘的产品，小霸王、金字塔也于 10 月份推出了仿 101 键盘的 586 和 9588 产品；金字塔在 94 年率先推出了外接中华机 140K 容量的 5·25 英寸软盘驱动器之后，95 年新春伊始裕兴又推出了与微机完全兼容的 3·5 英寸标准软驱。从应用软件来看，发展更加迅速，已推出了小学、中学乃至高中的各年级辅助教学软件，特别是裕兴推出的仿“金山北大 WPS6.0”的“裕兴 WPS 文字处理系统”，不仅操作与微机一致，而且文件可与微机相互交流。从而，使学习机的档次又升一级而迈进了电脑阶

层，使它变成了名符其实的“家庭电脑”(Family Computer)，完全可以充当现代家庭管理、办公自动化管理的廉价微机。

为了使电脑初学者能够迅速入门，进而登堂入室。我们依据循序渐进的原则，编写了这一套丛书。本套丛书分：

操作入门
文字处理
游戏 BASIC 语言
中文 BASIC 语言
LOGO 语言
汇编语言
电脑作曲
电脑绘图
BASIC 语言程序集
硬件与维修

共十册。

本书是丛书的第六册。以“指令寻址方式”、“指令功能分类”两条线，系统地介绍了 6527 汇编语言的指令系统。讨论了没有配备汇编语言工作系统的电脑学习机，如何在 F BASIC 工作状态下编写机器语言程序的方法和技巧；提供了 6527 反汇编、小监控工具程序清单和使用方法；列出了可被用户调用的系统子程序清单和调用条件；并以“真正的电子钟”软件编写为例，介绍了 F BASIC 语言与机器语言的联合编程方法。

本书内容详实、深入浅出，既适合初学者自学，也可作为编写任天堂游戏软件、电脑学习机应用软件的培训教材使用。

在编写过程中，陈建宇、韩玉文、李观勇、于勇、于腾潇等同志作了大量的工作，全书由中国地质大学苏子栋教授审

定，在此一并表示感谢。

编　者

丛书学习说明

本丛书是面向电脑初学者的科学普及读物。丛书的编写结构安排，从初学者最易接受的角度开始，顺着初学者的学习兴趣循序渐进、逐步深入，引导初学者在不觉中步入电脑的殿堂。

丛书的前五册为基础篇，重点介绍了电脑基础知识、电脑学习机的选购常识以及键盘、软盘驱动器、打印机等微机外设的基础知识；常用汉字输入法和中文文字处理操作方法；F BASIC、中文 BASIC、LOGO 三种计算机语言的指令及编程方法。

丛书的后五册为提高篇，重点讨论了 6527CPU 工作系统的特点、汇编语言的指令及编程方法；音乐板、PLAY 指令、机器语言三种电脑作曲方法；键控、BASIC 指令、机器语言三种绘图方法；精选了 60 个 F BASIC、中文 BASIC 语言程序；最后讨论了电脑学习机的硬件结构原理和基本维修方法。

整套丛书内容连贯，但又独立成章。十本书读者可依序学习，也可在学习了丛书之一《操作入门》后，根据自己的兴趣和爱好挑选学习。如需要进行文字处理的用户，只要再学习丛书之二就可胜任。爱好音乐的朋友可继续学习丛书之七，就可令电脑演奏音乐。若要进一步提高品位，还须学习丛书之五，使用机器语言编程才能演奏美妙的旋律；电脑绘图爱好者，则要在丛书之三、之八上下功夫。许多朋友在学习了游戏 BASIC 语言后，想自己动手编写任天堂游戏程序及各种实用程序，那么丛书之九提供了各种类型的程序范例供你参考。

本丛书的撰写宗旨就是辅导读者掌握电脑的基础知识，有深造要求的朋友，请继续参阅任天堂游戏理论书籍《F BAISC 语言与编程技巧》、《电脑游戏机硬件与编程特技》、《任天堂游戏编程探密》。

目　录

第一章　概述……（1）

第一节　数和进制……（1）
第二节　逻辑电路与逻辑运算……（17）
第三节　触发器与数据存储……（24）
第四节　计算机语言……（26）
第五节　汇编语言概述……（32）
第六节　电脑学习机硬件系统简介……（39）

第二章　6527 的寻址方式……（52）

第一节　6527 指令系统的特点……（52）
第二节　立即寻址……（54）
第三节　绝对寻址……（61）
第四节　零页寻址……（73）
第五节　隐含寻址……（79）
第六节　累加器寻址……（83）
第七节　绝对 X 变址寻址……（85）
第八节　绝对 Y 变址寻址……（88）
第九节　零页变址寻址……（90）
第十节　间接寻址……（93）
第十一节　相对寻址……（95）
第十二节　先变址间接寻址……（100）
第十三节　后变址间接寻址……（103）

第三章　6527 的指令系统 …… (106)

第一节　数据传送类指令 …… (106)
第二节　算术运算类指令 …… (117)
第三节　逻辑运算与位测试类指令 …… (125)
第四节　程序控制类指令 …… (129)

第四章　汇编与 BASIC 联姻 …… (134)

第一节　机器语言程序的写入与运行 …… (134)
第二节　6527 反汇编 …… (141)
第三节　6527 小监控系统 …… (144)
第四节　汇编与 BASIC 程序结合实例 …… (150)

第五章　简单程序和分支程序设计 …… (157)

第一节　简单程序设计 …… (157)
第二节　分支程序设计 …… (167)

第六章　循环程序设计 …… (178)

第一节　循环程序的结构与特点 …… (178)
第二节　数据块的传送 …… (180)
第三节　数据的查找 …… (187)
第四节　判断数据的类型 …… (188)
第五节　数据的累加 …… (192)

第七章　子程序设计 …… (195)

第一节　子程序的概念 …… (195)
第二节　单个子程序的设计 …… (197)

第三节　多个子程序的设计 …… (204)

第八章　堆栈程序设计…… (209)

第一节　现场的保护与恢复 …… (209)
第二节　排序程序的设计 …… (214)
第三节　乘除运算程序的设计 …… (222)

第九章　系统资源的利用…… (228)

第一节　系统对 RAM 的使用 …… (229)
第二节　系统子程序简介 …… (237)
第三节　电子钟程序设计 …… (251)
第四节　电子钟程序的优化 …… (261)

附录　6527 指令系统表 …… (267)

第一章　概　　述

电脑学习机的各个系统软件(游戏 BASIC、中文 BASIC、LOGO 等)和应用软件、所有的任天堂游戏软件等都是使用 6527 机器语言编写的。欲了解这些系统和各种软件是怎样工作的,如何有效地使用它们提供的丰富资源编写高效的应用程序、提高自己的编程水平,就必须学好 6527 机器语言。

机器语言全部由数字组成,不利于记忆和识别,编程效率较低。于是人们又发明了机器语言的过渡语言——汇编语言。

实际上,汇编语言与机器语言是一个问题的两个方面。同机器语言相比,汇编语言只不过更方便人们编写、调试程序罢了。因此,可以说:学习汇编语言就是为了更全面、更有效地掌握机器语言。

本章将从计算机语言的种类出发,介绍汇编语言的程序形式、语句格式、指令特点等基本概念;讨论 6527CPU 工作系统的硬件环境,系统寄存器、软开关的分类和使用,系统的中断和复位模式;讨论数据的进制关系;简单介绍基本逻辑电路和基本逻辑运算的特点等。

第一节　数和进制

一、数和进制的概念

"数"和"进制",大家并不陌生。日常生活中使用最多的是

十进制数，用 0～9 这十个数字，可以组合出无数个大大小小的数据，如 325、1995、9600000 等等。

概括地说：十进制数以 0～9 十个数字为基数，遵循由低位向高位“逢十进一”的原则。每个人都很习惯十进制的计数方法。

另外，我们还熟悉 7 进制（每个星期 7 天，逢七进一周）、12 进制（每年 12 个月，逢十二进一年）、24 进制（每天 24 小时，逢二十四进一天）、60 进制（1 小时 60 分钟、1 分钟 60 秒，逢六十进一）、100 进制（1 平方米等于 100 平方分米、1 平方分米等于 100 平方厘米，逢百进一）、1000 进制（1000 米等于 1 公里、1000 克等于 1 公斤，逢千进一）等等。这些都是我们司空见惯的进制计数方法，不觉为奇。那么，计算机中采用的是什么样的进制呢？

二、计算机中的计数方法

在电子计算机中，通常采用二进制、八进制和十六进制计数法。

1. 二进制数

二进制计数法最简单，它只有“0、1”两个计数基数，由低位向高位的进位原则是“逢二进一”。如十进制中的数字“2”在二进制中将表示为“10”，十进制的“3”将表示为“11”等。

2. 八进制数

八进制数由“0～7”八个基数组成，遵循“逢八进一”的原则。如十进制中的“8”在八进制中将表示为“10”、十进制中的“12”在八进制中表示为“14”等。

3. 十六进制数

十六进制数以“0～9、A～F”十六个基数组成，遵循“逢十

六进一”的原则。如十进制中的“11”表示为“B”、十进制中的“20”表示为“14”等。

4. 各进制间数据的对应关系

二、八、十、十六进制数据对应关系见表 1－1。

表 1－1　二、八、十、十六进制数据对应关系一览表

十进制	二　进　制	八进制	十六进制
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
50	110010	62	32
100	1100100	144	64
1000	1111101000	1750	3E8

5. 各进制数据的区别

为了简化讨论中各种进制数据的叙述，统一约定对数据采取加后缀的方法予以进制区别。对于

二进制数后面加“B”(Binary)

八进制数后面加“C”(Octal)

十进制数后面加“D”(Decimal)

十六进制数后加“H”(Hexadecimal)

一般十进制数据可以省略后缀，但其它进制的数据不能省略。

三、二进制数的优、缺点

为什么电子计算机偏偏采用二进制而不是其它进制呢？这是因为二进制只有两个基数，便于使用物理元、器件的两种不同稳定状态来表示。例如：电位的高、低，开关的通、断，电灯的亮、灭，脉冲的有、无等，都可以用来表示二进制数“1”和“0”。显然，这种物理元件在技术上是较易实现的。假若采用十进制数字，就要求一种有十个不同稳定状态的物理元件，制造这种元件将是十分困难的。即使制造出来，结构也必然很复杂，因此导致计算机的体积十分庞大。另外，二进制数的运算规则比较简单，这也是电子计算机采用二进制数的一个重要原因。

但是二进制数的位数较多，书写和识读都很不方便。例如十进制的 15 仅有两位，若写成二进制数则为四位(1111)；而 117 的二进制形式将为七位(1110101)。另外人们也不习惯使用二进制数据进行计算。

为了解决二进制数位多、识读困难的矛盾，一般在编程中使用八进制或十六进制数据，仅在输入计算机后才转换为二进制数据的形式。电脑学习机中仅使用十六进制。

四、不同进位计数制之间数据的转换

1. 数位的权

对于十进制数“1234”，“4”为个位(10的0次方)、“3”为十位(10的1次方)、“2”为百位(10的2次方)、“1”为千位(10的3次方)。若用数学语言描述则是：“4”的“权”是10的0次方、“3”的权是10的1次方、“2”的权是10的2次方、“1”的权是10的3次方。因此，任何一个十进制数都可以表示成它的数字符号与“权”相乘后的组合，即分解成10的方次和的形式。如：

$1234=1\times10^3+2\times10^2+3\times10^1+4\times10^0$

相应地十进制小数部分的“权”依次为

10^{-1}　　10^{-2}　　10^{-3}　　……

同样，二进制整数各位的“权”从低到高，依次为

2^0　　2^1　　2^2　　2^3　　2^4……

小数部分的“权”由左向右依次为

2^{-1}　　2^{-2}　　2^{-3}　　2^{-4}　　2^{-5}　　……

如二进制数“111.01”可表示为

$101.01=1\times2^2+0\times2^1+1\times2^0+0\times2^{-1}+1\times2^{-2}$

2. 十进制数转换为二进制数

在各数制的转换中，整数部分仍然转换为整数、小数部分仍然转换为小数。

(1)整数的转换

十进制整数转换为二进制的方法是：将十进制整数不断地除以 2，直到商等于 0 为止；然后以相反的顺序取余数，即得到对应的二进制整数。如：

```
2|155
 2|77    ……………… 余数为1
 2|38    ……………… 余数为1
 2|19    ……………… 余数为0
 2| 9    ……………… 余数为1
                   余数为1
  2|4    ……………… 余数为1
  2|2    ……………… 余数为0
  2|1    ……………… 余数为0
    0    ……………… 余数为1
```

从底向上取余数依次为“10011011”，即

155D＝10011011B

(2)小数的转换

十进制小数转换为二进制小数的方法是：将十进制的小数部分不断地乘以 2，顺序取整数，直到小数部分等于 0 为止。如：

```
    0.8125
×        2
----------
    1.6250      ………………整数部分为 1
```

$$\begin{array}{r} 0.625 \\ \times \quad 2 \\ \hline 1.250 \end{array}$$ ……整数部分为 1

$$\begin{array}{r} 0.25 \\ \times \quad 2 \\ \hline 0.50 \end{array}$$ ……整数部分为 0

$$\begin{array}{r} 0.5 \\ \times \quad 2 \\ \hline 1.0 \end{array}$$ ……整数部分为 1

顺序取整数为“1101”，有

$$0.8125D=0.1101B$$

对于无限乘下去，始终不能达到小数部分等于 0 的情况，可视精度要求选取足够的位数。

(3)包含整数、小数部分数据的转换

对于既含整数部分又含小数部分的十进制数据转换为二进制的方法是：将十进制数的整数、小数部分分别转换为二进制的整数、小数后，再用小数点把整数、小数部分连接起来。如

$$155.8125=10011011.1101B$$

3. 二进制数转换为十进制数

根据二进制数据的定义，把二进制数按“权”展开相加后就得到对应的十进制数据。如：

$$\begin{aligned} 101.11B&=1\times2^2+0\times2^1+1\times2^0+\\ &\quad 1\times2^{2-1}+1\times2^{-2}\\ &=4+0+1+0.5+0.25\\ &=5.75 \end{aligned}$$

即

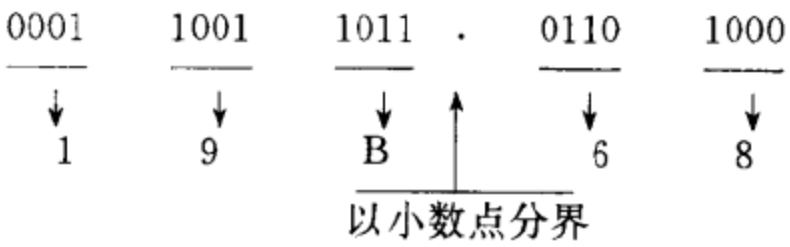
$$101.11B=5.75D$$

4. 二、十六进制数之间的转换

由于2的4次方等于16，故可以使用1位十六进制数表示4位二进制数。这样，一个字节(8位)的二进制数就可以用2位十六进制数表示出来。

(1)二、十六进制数据转换

二进制数据向十六进制数据转换的方法是：以小数点为分界点，以4位为单位分别向左、右分组，不足4位的补"0"，然后依据表1－1，把每组的二进制数分别用十六进制数表示即可。如110011011.01101B转换为十六进制数时的分组：

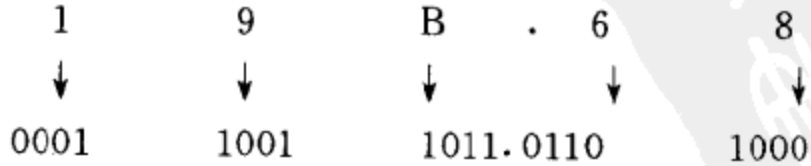


所以

110011011.01101B＝19B.68H

(2)十六、二进制数据转换

十六进制数据向二进制数据的转换方法是：依据表1－1，把每位16进制数据分别用对应的4位二进制数表示；然后把这些二进制数连接起来即可。如19B.68H转换为二进制数据时的操作如下：



所以

19B.68H＝000110011011.01101000B

删去数据头、尾的零后，得

19B.68H=110011011.01101B

5. 十、十六进制数之间的转换

(1)十进制数转换为十六进制数

整数转换的方法是：用十进制数据反复除以 16，直到商等于零为止；再反序取余数。

小数的转换方法是：用十进制数据的小数部分反复乘以16，顺序取整数，直到小数部分等于零为止。若遇到无限相乘、小数部分始终不等于零的情况可视精度要求截取一定的位数。

对于既有整数部分又有小数部分的数据，可把两部分分别转换后，再加小数点将两部分连接。

如 1288.125D 转换为十制数的操作过程为：

首先转换整数部分，有

```
16|1288
  16|80    ………………  余数为8
    16|5   ………………  余数为0
       0   ………………  余数为5
```

再转换小数部分，有

```
     0.125
×       16
      750
     1 25
     2.000   ………………整数部分为 2
```

所以

1288.125D=508.2H

(2)十六进制数转换为十进制

方法是：按十六进制各位的权将数据展开，取其相加和即为十进制数据。如

$$508.2H=5\times16^{2}+0\times16^{1}+8\times16^{0}+2\times16^{-1}$$
$$=1280+0+8+0.125$$

五、二进制数的运算规则

电子计算机中，数的运算以二进制的算术运算为基础，运算规则非常简单，下面简单介绍它的四则运算。

1. 加法

二进制的加法规则是：

0+0=0

0+1=1

1+0=1

1+1=10

例 1—1 求 38+10=？

	十进制	二进制
被加数	38	100110
加数	+ 10	+ 1010
和	48	110000

2. 减法

二进制的减法规则是：

0−0=0

1−0=1

1−1=0

$10-1=1$

例 1—2　求 $48-38=?$

	十进制	二进制
被减数	48	110000
减数	− 38	+100110
差	10	001010

3. 乘法

二进制的乘法规则是：

$0\times0=0$

$0\times1=0$

$1\times0=0$

$1\times1=1$

例 1—3 求 $14\times5=?$

	十进制	二进制
被乘数	14	1110
乘数	× 5	× 101
积	70	1110 0000 1110 1000110

从上例可见，做二进制乘法时，若乘数为 1，则被乘数照写一遍，其最右边的一位应与相应的乘数对齐；若乘数为零，

实际上没起什么作用(仅控制下一个位积向左移一位)。乘数的各位运算完毕后,把各位积相加即求得结果。可见,二进制乘法的运算过程是由移位和加法两类操作实现的。

4. 除法

二进制除法是乘法的逆运算,它是由减法和移位两类操作实现的。

例 1—4 求 70÷5=?

```
      十进制                二进制

                                 1110
        14                   ________
     ____               101 | 1000110
  5 | 70                        101
      5                     ---------
     ----                       11110
      20                        101
      20                     --------
     ----                        1010
       0                         101
                              -------
                                    0
```

总之,二进制的四则运算可归结为加法、减法和移位三种操作。实际上,在计算机中并不直接进行减法运算,而是采用加负数来代替减正数的运算。这样一来,四则运算就简化为加法和移位两类操作了。

5. 二进制中的负数

在电子计算机中,怎样表示数的正、负,怎样用加负数的方法代替减法呢?

计算机中表示数的正负不是在数的前面加“+、-”号,而是在二进制数的前面增加一位“符号位”。当“符号位”为 0 时,表示此数为正;当“符号位”为 1 时则此数为负。例如在 8 位机

中，“+48、-38”的二进制数形式为：

+48=0.0110000B

-38=1.0100110B

其中圆点前面的“0、1”就是符号位，圆点后面为数据（注：圆点是为便于阅读和区别而人为添加的，实际上没有圆点。即以上两数在计算机内部分别表示为“00110000、10100110”）。

6. 字与字长

计算机中所有的信息都是以二进制代码的形式表示的，这个二进制代码就称为“字”，“字”的位数就称为“字长”。通常说的 8 位机、16 位机就是指它们所能处理的字的位数为 8 位、16 位。显然，计算机的位数越多、能表示数据的位数就越多，计算精度越高。但是，字长越长，计算机的结构和造价就越高。因此，中华学习机、电脑学习机都选择了 8 位机。

8 位机中，字长为 8 位。为讨论叙述方便起见，人们按位数的顺序，从低到高对每一位起一个名字 D0～D7，见下表

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	0	0	0	1

表中，D7、D5、D4、D0 位为 1，D6、D3、D2、D1 位为 0。

显然，对无符号数来说，8 位机每个字所能表示的数据范围为 0～255（0～FFH）；对有符号数所能表示的数据范围为 +127～-128。

7. 原码、反码和补码

计算机减法运算中，加负数是通过加补码的方法实现的。什么叫“补码”呢？

对于一个二进制负数来说，可以有三种不同形式的编码，即原码、反码、补码。

(1)原码

原码保持了二进制数的原来形式，即数的数据部分不变，只是符号位根据数的正、负取“0”或“1”。

(2)反码

反码就是将数的数据部分按位取反，即“0”代换为“1”、“1”代换为“0”。如“－38”的二进制原码为“1.0100110”，其反码就为“1.1011001”。

(3)补码

补码就是在反码的最低位加1，如“－38”的补码为“1.1011010”。

(4)三类码的表示方法与应用

若引入变量X，设X的原码记为$[X]_{原}$、X的反码记为$[X]_{反}$、X的补码记为$[X]_{补}$，则有

$X=-38$

$[X]_{原}=1.0100110$

$[X]_{反}=1.1011001$

$[X]_{补}=1.1011010$

下面以＋48－38为例说明怎样用加补码的方法实现减法。

＋48的二进制数原码　　0.0110000

－38的二进制数补码　＋1.1011010

和　　10.0001010

对于8位机，符号位为最高位，再往前的进位由于没有硬件保证而丢失。因此，上例的计算结果为

0.0001010

因符号位等于0，所以结果是正数，即十进制的10。运算

结果显然是正确的。

当两数相减的结果为负数时，上述方法仍然适用，只不过求出的结果为补码，应将其变为原码才能得到正确的结果。

例 1－5 求 38－48＝？

＋38 的二进制原码		0.0100110
－48 的二进制补码	＋	1.1010000
结果为二进制补码		1.1110110

符号位等于 1，结果为负数。

已知补码求原码，是已知原码求补码的逆运算，必须将补码减 1 后再按位取反。减 1 的操作仍可用加 1 的补码实现。则有

结果的二进制补码		1.1110110
－1 的二进制补码	＋	1.1111111
结果的二进制反码		11.1110101
把反码变为原码		1.0001010

所得结果为“1.0001010”，即十进制数“－10”。

实际上，已知补码求原码，也可以按各位取反再加 1 的方法求得。如上例

结果的二进制补码	1.1110110
结果的二进制反码	1.0001001
结果的反码加 1 等于	1.0001010

与补码减 1 求反码的结果相同，请读者自己分析原因。

8. 进位与溢出

从以上讨论可知，运算中多数情况要涉及补码运算，而在补码运算中将涉及到进位和溢出两个概念。

(1)进位

在8位机中,进位是指码值超过8位的情况。当运算结果大于255(11111111B)时就要产生进位。在补码运算中,丢掉进位后的结果是正确的。

(2)溢出

在8位机中,溢出是指带符号数的取值超出了-128～+127的数据范围。当取值超出这个范围时,就产生溢出,运算结果将出错。

例1-6 计算-65-64=?

$$\begin{array}{ll} -65\text{的补码} & 10111111 \\ -64\text{的补码} & \underline{+\quad 11000000} \\ & 101111111 \end{array}$$

运算结果为+127,显然是错误的,这是因为

-65-64=-129

超出了取值范围产生溢出造成的。

当运算结果产生溢出时,应停止运行,分析产生错误的原因,纠正错误。

运算结果有无溢出,可按如下方法判断:

设X0为数X的符号位、Y0为数Y的符号位、Z0为运算结果的符号位,若

X0=0、Y0=0、Z0=0,则无溢出;

X0=1、Y0=1、Z0=1,则无溢出;

X0=0、Y0=0、Z0=1,则有溢出;

X0=1、Y0=1、Z0=0,则有溢出。

以后的学习将知道,CPU标志寄存器中专有一位标志运

算结果的溢出情况。

关于进位与溢出的认识，将随着学习的深入而逐渐加深。当前只要具有初步的概念就可以了。

通过以上几例，读者可能觉得使用二进制数，运算次数太多、太麻烦、工作效率太低。诚然，若手工计算，用这种方法确实太笨。但这种"笨方法"的运算操作简单，而电子计算机最擅长的就是进行这方面的工作。因它每秒钟可以进行数百万次的运算，所以，它就可以"笨中取巧"，用最简单的操作完成最复杂的运算。

第二节　逻辑电路与逻辑运算

计算机中，通过什么样的电路实现二进制数据运算的呢？

如前所述，二进制数只有两个基本数字 0、1，对二进制数的运算实际上可归结为对 0 和 1 的运算。0、1 可以与逻辑关系中的"是、非"相对应，因此对 0、1 的运算也就是对是、非的运算，这一运算过程就是逻辑判断。电子计算机中，正是使用一些能够进行逻辑判断的电路来实现二进制数运算的。

能够对 0、1 进行基本逻辑运算的电路称为"基本逻辑电路"。常用的基本逻辑电路有：或门逻辑电路、与门逻辑电路、非门逻辑电路和异或门电路。从逻辑电路的状态抽象出的二进制运算规则，就称为逻辑运算。因此，相应地运算有：逻辑或、逻辑与、逻辑非、逻辑异或。

一、或门逻辑电路

1. "或"现象电灯电路

我们从人们最熟悉的电灯电路开始，见图 1－1。

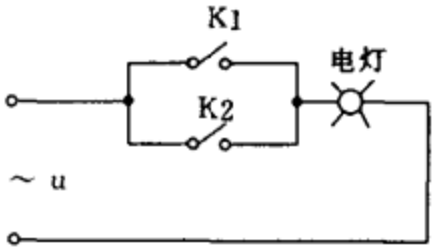


图 1－1 “或”现象电路

在图 1－1 中，开关 K1 和 K2 并联后再串接于电源和负载（电灯）电路中。如果约定灯亮为“1”状态、灯灭为“0”状态，那么开关的闭、合与电灯的亮、灭有表 1－2 的对应关系。

表 1－2 “或”现象开关、电灯状态一览表

开关状态		电灯状态	说　　明
K1	K2		
0	0	0	K1、K2 都断开，灯灭
0	1	1	K1 断、K2 合，灯亮
1	0	1	K1 合、K2 断，灯亮
1	1	1	K1、K2 都合上，灯亮

2. 逻辑“或”运算关系式

如果把开关状态称为输入、电灯状态称为输出，则表中所列四种现象代表的输入、输出之间的关系就可以表示为：

0＋0＝0

0＋1＝1

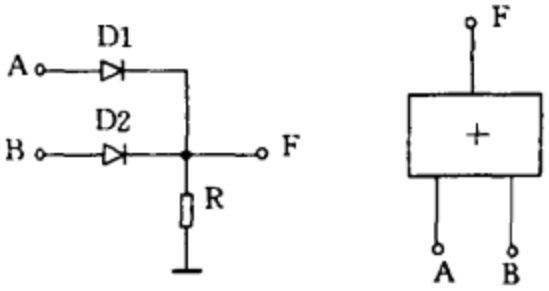
1＋0＝1

1＋1＝1

以上等式就是逻辑加法（逻辑或，记为 OR）的关系式。这与二进制加法规则是很相近的，除“1＋1＝10”与逻辑加法中的“1＋1＝1”不同外，其余三式完全相同。

3.“或”门电路

在电子计算机中，能够实现逻辑加法的电路称为“或门”逻辑电路，简称“或门”。图 1－2 所示的就是最简单的或门电路。



(A)电路图　　(B)符号图

图 1－2　或门逻辑电路

图 1－2(A)中，D1、D2 为晶体二极管，A、B 为输入端，F 为输出端，R 为输出电阻。约定：输入端 A、B 为高电平时代表“1”状态，低电平则代表“0”状态。同样，输出端 F 为高电平时代表输出为“1”状态，低电平为“0”状态。当 A、B 中有一端为高电平时，电流通过二极管经电阻流向地，在电阻 R 上产生电压降，使输出端 F 为高电平；当 A、B 都是高电平时，F 也是高电平；只有 A、B 都是低电平时，电流被 D1、D2 阻挡，没有电流通过，R 上面没有压降，F 才为低电平。输出和输入的关系可用如下关系式表达：

$F=A+B$

输入端的数目可以超过 2 个，如 3 个、4 个或更多，其关系式可写为

$F=A+B+C+D+\cdots\cdots$

在电子计算机的逻辑图中，或门逻辑电路常用图 1－2(B)所示的符号表示，矩形框内的“＋”号表示逻辑“或”。

二、与门逻辑电路

1.“与”现象电灯电路

图 1－3 示出了“与”现象的电灯电路。

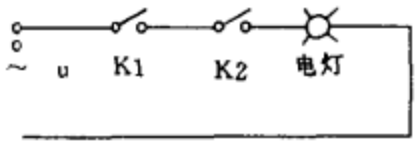


图 1－3 “与”现象电路

表 1－3 “与”现象开关、电灯状态一览表

开关状态		电灯状态	说明
K1	K2		
0	0	0	K1、K2 都断开，灯灭
0	1	0	K1 断、K2 合，灯灭
1	0	0	K1 合、K2 断，灯灭
1	1	1	K1、K2 都合上，灯亮

2. 逻辑“与”运算关系式

“与”门逻辑电路输入、输出的对应关系如下：

$0\times0=0$

$0\times1=0$

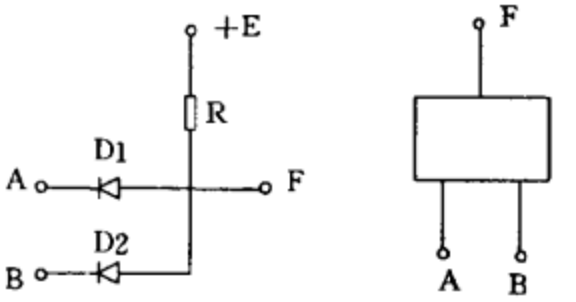
$1\times0=0$

$1\times1=1$

这就是逻辑乘法(通常称为逻辑与，记为 AND)，与二进制乘法的规则完全相同。

3.“与”门电路

在电子计算机中，能够实现逻辑乘法的电路称为“与门”逻辑电路，简称与门。最简单的与门电路如图 1－4 所示。



(A)电路图　　(B)符号图

图 1－4　与门逻辑电路

三、非门逻辑电路

1. “非”现象电灯电路

非门逻辑电路可用图 1－5 所示的电灯电路说明。

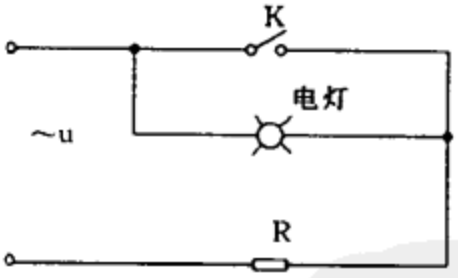


图 1－5　“非”现象电路

开关状态与电灯状态的对应关系见表 1－4。

表 1－4　“非”现象开关、状态一览表

开关状态	电灯状态	说明
0	1	K 断开，灯亮
1	0	K 闭合，灯灭

2. 逻辑“非”运算关系式

由图 1－5、表 1－4 可见，当 K 断开时，电流经过电灯、电

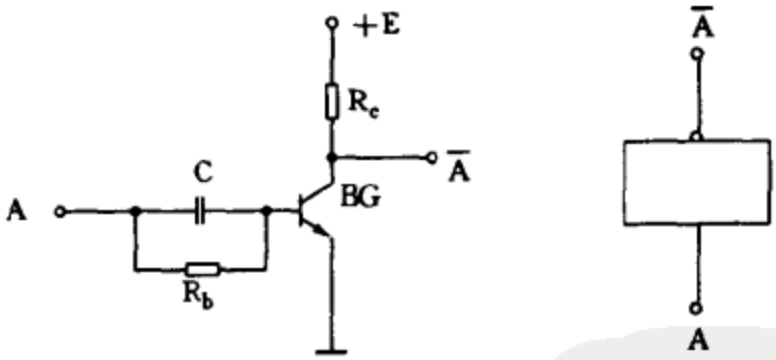
阻构成的回路，电灯亮；当 K 闭合时，电灯被开关短路，电流经开关、电阻构成的回路，电灯两端无电位差，故电灯灭。这里的开关 K 起到对电灯亮、灭的否定作用，一般称为“非”。若输入状态记作“A”，则输出状态记作“$\overline{A}$”，读作“非 A”。

逻辑“非”运算很简单，其关系式为：

$\overline{0}=1$　　$\overline{1}=0$

3.“非”门电路

电子计算机中，能够实现“非”这一逻辑关系的电路称为非逻辑电路，简称非门，通常又叫作“反相器”。图 1－6 为反相器的一种典型电路。



(A)电路图　　(B)符号图

图 1－6　非门逻辑电路

当输入端为高电平时，晶体三极管导通，电流由＋E 流经 Rc 和 BG 到地，由于 BG 导通时内阻极小，输出端 $\overline{A}$ 为低电平，即若 A 为“1”，$\overline{A}$ 为“0”。反之，若 A 为“0”则 $\overline{A}$ 为“1”(A＝0 时，BG 截止，＋E 通过 Rc 使输出端为高电平)。在这里，BG 起着开关的作用。非门的表示符号如图 1－6(B)所示，矩形框上面的小圆圈表示“非”。

四、异或运算

1. “异或”现象电灯电路

“异或”逻辑电路可用图 1－7 所示的双联开关电灯电路表示。

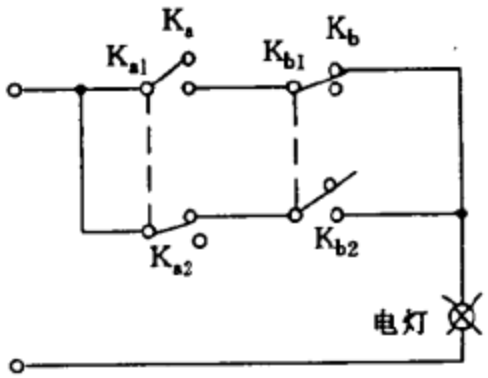


图 1－7　异或现象电路

图 1－7 中，开关 Ka、Kb 分别为双联开关（如扳动 Ka 时，Ka1、Ka2 同时动作），它们的向上、向下动作与电灯的亮、灭关系如下：

当两个开关都向上时，Ka1 断、Kb1 合、Ka2 合、Kb2 断，电灯灭；

当两个开关都向下时，Ka1 合、Kb1 断、Ka2 断、Kb2 合，电灯灭；

当 Ka 向上、Kb 向下时，Ka1 断、Kb1 断、Ka2 合、Kb2 合，电灯亮；

当 Ka 向下、Kb 向上时，Ka1 合、Kb1 合、Ka2 断、Kb2 断，电灯亮。

可见，两个开关的方向相同时电灯灭，方向不同时电灯亮。如果约定开关向上为“0”、向下为“1”，则开关状态与电灯

亮灭的对应关系如表1－5所示。

表1－5 “异或”现象开关、电灯状态一览表

开关状态		电灯状态	说　　明
K1	K2		
0	0	0	Ka、Kb都向上，灯灭
0	1	1	Ka上、Kb下，灯亮
1	0	1	Ka下、Kb上，灯亮
1	1	0	Ka、Kb都向下，灯灭

2. 逻辑“异或”运算关系式

异或运算的关系式如下：

$0 \odot 0 = 0$

$0 \odot 1 = 1$

$1 \odot 0 = 1$

$1 \odot 1 = 0$

关系式中“⊙”为异或运算符号。从关系式可见，异或运算的特点是：两状态相同结果为0；两状态不同结果为1。

“异或”也是逻辑运算中经常用到的一种运算方式，由于其逻辑电路较复杂，不再画出。

第三节　触发器与数据存储

触发器是一种存储器件，它可以以低电平或高电平的方式存储一位二进制数据。

1. 与非门

一个与门和一个非门串接在一起就组成了一个复合门电路，称为“与非门”。与非门的逻辑功能为：只有当输入端均为高电平时，输出端才为低电平；否则，输出为高电平。与非门的

符号如图 1－8 所示。

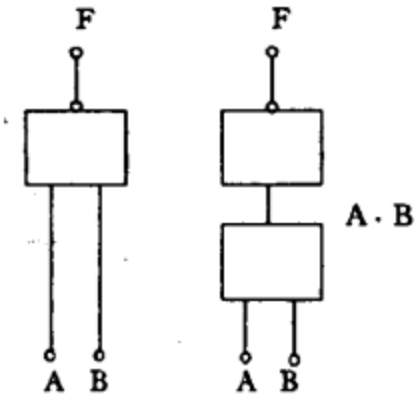


图 1－8　“与非”门符号

2. R－S 触发器

用两个与非门可以组成一个最简单的触发器，称为 R－S 触发器，如图 1－9 所示。

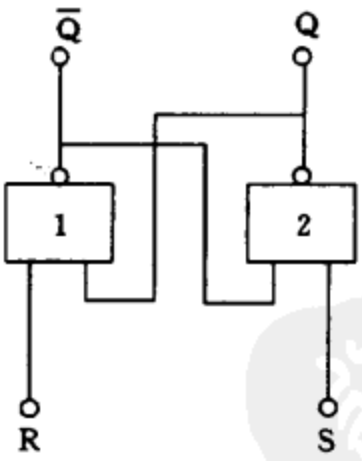


图 1－9　基本 R－S 触发器

图中 R 称为置“0”输入端、S 称为置“1”输入端。在没有脉冲输入时，R、S 端均为高电平。R－S 触发器有两个输出端，即 Q 和 $\overline{Q}$ 端，两个输出端互为反相。当 Q 端为高电平时，$\overline{Q}$ 端为低电平；反之，当 Q 端为低电平时，$\overline{Q}$ 端为高电平。

3. R－S 触发器的工作原理

(1)当没有触发脉冲输入时，R＝“1”、S＝“1”，即 R、S 均

为高电平。假定这时 Q＝“1”、$\overline{Q}$＝“0”。

(2)当 R 端有触发脉冲时，即 R＝“0”、S＝“1”，由与非门的特性可知，$\overline{Q}$＝“1”。这时与非门 2 的两个输入端均为“1”，则 Q＝“0”，所以说 R 是置“0”端。现在，即使 R＝“1”，也仍保持 Q＝“0”的状态。

(3)当 S 端有触发脉冲时，R＝“1”、S＝“0”，与非门 2 的一个输入端为低电平，故必然有 Q＝“1”。现在与非门 1 的两个输入端均为高电平，故有 $\overline{Q}$＝“0”。当前即使 S 端的触发脉冲消失，也仍然保持这种状态，所以把 S 称为置“1”端。

可见，只要控制 R、S 端的触发脉冲，就可以改变触发器的输出端电平，即控制触发器储存“0”或“1”。不难看出，一个 R－S 触发器可以储存一位二进制数。

从以上两节的讨论可见，与门、或门、非门、异或门是判定器件，使用它们可以完成算术运算或逻辑运算。触发器是存储器件，使用它可以组成存储器，以把各种数据、信息储存起来。有了运算、存储器件，若再加上一些辅助电路和设备就可以组成电子计算机了。

总而言之，电子计算机虽然复杂而神奇，但归根结底，它只能运算、储存 0 和 1 两个数字，而它们的基本电路也非常简单。问题在于有了这些积木块，就看你如何使用它们去建造小别墅或摩天大楼了。

第四节　计算机语言

一、语言的概念

“语言”——大家非常熟悉，我们天天说、天天用，张口就

来、提笔即写，似乎不须多讲。但若要求对语言下一个确切的定义，则不是随口就来的事了。

“语言”是人类进行思想交流的工具。概括地说，它是由词汇和语法组成的“符号”系统。如

“我们是中国人”

就是由词汇“我们、是、中国、人”四个词汇按照“主、谓、宾”等语法结构组成的一句话（一串符号）。这里的词汇可以概括地称为“符号”，不同的符号和语法就可以组成不同的语言，如汉语、英语、日语、法语等。哑语、旗语的符号是由动作组成的，它是符号的特殊形式。

通常，人们把这样的符号系统称为自然语言，这些语言已为人们所熟悉、所理解、所掌握、所应用，在人类的相互交流中起到了巨大的作用。可以说，没有语言，人类之间就没有交流。

但是，电子计算机不能识别自然语言，它所能识别、处理的是它自己的一套独立的语言——计算机语言。也就是说：人与电脑间的交流只能通过计算机语言才能实现。这就如同中国人要与英国人交谈必须学习英语一样，我们必须学习计算机语言。

那么，计算机语言的结构是怎样的呢？

二、计算机语言的形式和类别

计算机语言有高级语言、低级语言之分：高级语言有许多种；低级语言则分为机器语言和汇编语言。

机器语言是计算机的心脏 CPU（中央处理器）所唯一能够识别和直接执行的语言。也就是说，用其它语言编写的程序最终要翻译成机器语言的形式，计算机才能够执行。

1. 机器语言

机器语言是由 CPU 能够直接执行的机器代码所构成的语言。使用机器语言的优点是:编写的程序不须翻译,CPU 即可直接执行。因此,它具有识别效率高、执行速度快等优点。

但是,机器语言使用的"词汇"——机器代码,是二进制数据。而二进制数据仅有"0、1"两个数码,显然用机器代码书写程序非常繁琐且极易出错。例如:将内存单元 $60、$61 中的数值相加("$"表示后面的数据是十六进制数形式的地址序号),结果存入 $62 单元中,其 6527 机器语言程序为:

10100101
01100000
00011000
01100101
01100001
10000101
01100010

可见,这种语言存在着严重的缺点:

其一是二进制数据看起来都差不多,尤其在大量阅读程序时,令人眼花缭乱,难以区别;

其二是二进制码必须一个接一个顺序输入计算机,稍不留心就会出错,且输入速度缓慢;

其三是人们很难迅速理解电脑所要完成的是何种任务,执行的是何类操作;

其四是程序冗长,编写麻烦,容易混淆,很难发现错误。即使发现错误纠正也不方便;

其五是记忆困难,人们很难一一记住所有指令的机器代码、数据单元地址以及所有的指令地址。

由于十六进制数据与二进制数据有着简单的换算关系,

所以为使程序书写简洁，人们采取了十六进制数形式来表示二进制代码，这样上例的程序就可以写为：

A5 60

18

65 61

85 62

这种书写形式简洁多了。

但是，采用了十六进制数后，人们仍然难以区别何者是指令、何者是数据、何者是地址；也难以知道程序在作什么，将到哪里去；不易区分“A5 65”各代表什么意思；而且不同类型的微处理器，其代码的含义还有所不同。所以，使用十六进制数的形式仍然不能解决机器语言阅读、理解困难的问题。因此，人们继续探求比较接近自然语言的计算机语言来编写程序，从而出现了汇编语言。

2. 汇编语言

汇编语言也叫符号语言，它是一种面向机器的程序设计语言。它的最大特点是用符号代替机器码，从而使程序的书写比机器代码方便许多。为便于记忆，汇编语言把所有的机器操作码都以英文单词（或缩写）和符号的形式进行表示，这些单词和符号称为“助记符”（取帮助记忆之意），助记符可以反映指令的功能和主要特征。上例程序用助记符表示的形式为：

LDA $60

CLC

ADC $61

STA $62

这个程序的功能虽然还不够明显，但其中某些部分的功能已变得较容易理解了。如上述四条指令的含义、功能如下：

第一条指令“A5”的助记符是“LDA”，它是英文“LoaD Accumulator”(装载累加器)的缩写，“累加器”是 CPU 中的一个寄存器，记为“A”。指令

LDA $60

的意思是“取地址为 $60 存储单元中的数据送入寄存器 A”；

第二条指令“18”的助记符为“CLC”，它是英文“CLear Carry”(清除进位标志位)的缩写。作用是“清除标志寄存器中进位标志位”(置进位 C=0)；

第三条指令“65”的助记符为“ADC”，它是英文“ADd to accumulator with Carry”(连同进位标志位与寄存器 A 中的内容相加，结果送入 A 中保存)的缩写。指令

ADC $61

的含义是：取 $61 单元的数据与进位标志 C、累加器 A 中的数据相加，相加之和仍存入累加器 A 中。

第四条指令“85”的助记符为“STA”，它是英文“STore Accumulator to memory”(把累加器 A 中的内容送入存储器)的缩写。指令

STA $62

的含义是把累加器 A 中的数据送入存储器 $62 单元。

可见，在汇编程序中，我们已经知道哪部分是指令、哪部分是数据、哪部分是地址了。

我们知道，计算机只能理解和执行二进制数代码的机器语言程序，因此汇编语言程序编写完毕后，只有通过编译软件将其还原为机器语言程序，才能供计算机直接执行。常用的编译软件有 EDASM(Editor/Assembler)和 LISA 等。

一般把汇编语言程序称为“源程序”，把经编译程序处理后的二进制代码程序称为“目标程序”。

汇编语言比机器语言前进了一大步，它具有形式直观、方便记忆等优点。但是由于汇编编程中，仍要考虑数据的读取、传送、存放、操作，仍要考虑内存的安排使用等，机器语言所具有的大部分缺点（如繁琐、易出错、不直观），汇编语言同样存在，所以它仍是一种面向机器的低级语言。

3. 高级语言

通常把接近自然语言的计算机语言称为高级语言。计算机高级语言是一种适合人的思维表达方式、接近英语自然语言的语言系统，其表达方式接近于被描述的问题。高级语言是由不同的“关键字”和“表达式”按照一定的语法规则组成的。如我们以上所学的 F BASIC 语言、浮点 BASIC 语言、LOGO 语言等。

目前，世界上已有数百种不同功能的计算机高级语言，其中得到广泛采用的有十几种。最常用的除 BASIC、LOGO 语言外，还有 FORTRAN、Pascal、COBOL、FoxBASE 语言等。

高级语言面向问题、面向使用者。用户编程时不必关心计算机的内部结构、数据流向和内存的使用情况，简单易懂，特别适合初学者学习、掌握。因此，它方便了程序的编写、阅读和交流，但由于它脱离机器运行环境太远，程序的运行速度将受到极大的影响。当电脑运行某些高级语言程序如 F BASIC 时，必须首先经过解释程序，把当前的操作翻译为可用二进制代码表示的形式后，CPU 才能执行，故程序的执行速度较慢。实际上 CPU 就像一个只认识汉语拼音的小男孩，如果给他用汉语拼音写的读物，他可以直接阅读；若给他用汉字书写的读物时，必须把汉字翻译成汉语拼音后，他才能读明白。这就是用 F BASIC 语言编写的游戏程序比任天堂游戏逊色许多

的主要原因。

第五节　汇编语言概述

一、为什么学习汇编语言

有人问：既然高级语言具有一系列的优点，为什么还要学习汇编语言呢？

这是由汇编语言的特点所决定的。因汇编语言程序并不是机器直接执行的程序，它仅是人们为了编程、修改方便而采取的一种符号系统，最终还要恢复到机器语言的形式，而一条可执行的汇编语言指令等效于一条机器语言指令，所以，机器语言所具有的特点，就是汇编语言的特点。

1. 程序结构紧凑，可最大限度地节省内存空间和 CPU 资源，程序效率高。

2. 汇编语言经编译后，电脑可以直接执行，程序的运行速度最快。

3. 能够准确掌握程序的执行时间，可充分利用系统的中断资源。

4. 方便实时控制、数据采集，特别适合工业自动化控制。

5. 适用于应用软件、系统软件的开发。

由此可见，汇编语言是高级程序员的工具，它是电脑编程进入高级阶段的必修课。

实际上，任何高级语言的功能，最终还是要通过执行机器语言才能实现。

二、汇编语言的语句格式

汇编语言程序由一系列语句组成，每个语句单独占一行。语句分两类：一类是在程序运行中执行操作的、由指令构成的指令语句；另一类是在程序运行中不产生操作的、由伪指令构成的非执行语句。它们都必须按语句规定的格式书写。

一行语句可以由四段组成，或者说分四个区：标号区、操作码区、操作数区、注释区。

语句的一般格式为：

标　号　　操作码　　操作数　　注　释

Label　　Opcode　　Operand　　Comment

四段的书写要求如下：

字段之间用空格作为分隔符；

同一字段内不允许出现空格；

操作数与注释之间用分号“；”隔开。

例如第四节的一段汇编程序可以这样书写：

```
标　号    操作码  操作数    注　　释
Label     Opcode  Operand   Comment
START     LDA     $60       ;将$60单元的数据送入累
                            加器A
          CLC               ;清除进位标志
          ADC     $61       ;累加器中的内容与$61单
                            元、进位标志
                            ;C相加，结果存在A中。
          STA     $62       ;将累加器A中的数据送入
                            $62单元
```

由示例可见，每行语句中，标号、操作数、注释三项不一定

同时具备，但操作码必不可少。

三、标号

语句中的标号以字符串的方式，表达语句行所在内存单元的地址。这种地址不是使用十六进制数据表示，而是使用符号、文字表示的，通常称为符号地址。

当某行指令之前冠以标号时，这个标号就代表该指令存储在程序存储区的地址值。当转换成目标程序时，标号将被具体的程序存储地址所取代。因此，编程时程序员不必算出存储单元的具体地址值，这就使程序单元的查找和替换变得十分容易，也给程序的编写和调试带来了很大的方便。

使用标号有以下要求：

1. 标号的位置

标号必须处于一个语句行的开始位置。它的符号组合必须是唯一的。即在一个源程序中，同名的标号只能使用一次，不允许重复使用。

2. 标号的组成

标号必须以英文大写字母开头，以 8 个字符为限，以空格作为结束符号。

3. 标号的禁忌

不允许使用操作码或助记符作为标号，也不能使用 6527CPU 内部寄存器的名字作标号。

4. 标号的省略

标号并不是程序行中所必需的，可根据编程的需要选用。一般在程序的入口地址、程序的转移地址、计数器单元、常数单元等处使用标号。

四、操作码

6527CPU 有 56 种基本指令，每条指令包含 1～3 个字节。除单字节指令外，每条指令都包含操作码、操作数两部分。

操作码是语句的必备部分。它必须放在指令的第一个字节中，用于指明操作的性质，规定当前指令进行的操作种类。

操作码由指令助记符或伪指令构成。指令助记符一律由三个字母构成，伪指令则由两个以上的字母构成。例如：

```
ORG  $8000        ;规定程序的起始地址为 $8000
LDA  #$FF         ;取立即数 FFH 到累加器 A
STA  $77          ;令 $77、$78、$79 单元等于 FFH
STA  $78
STA  $79
RTS               ;子程序返回
```

上述程序中，ORG、LDA、STA、RTS 都是操作码，其中 ORG 为伪指令，其余三个为指令助记符。

五、操作数

操作数是操作码所要进行操作的数据或地址。操作数是操作码的补充，它和操作码一起确定指令所要执行的内容。

在汇编语言中，操作数可以取标号、常数、表达式、字符或字符串等多种形式。

1. 标号形式的操作数

标号作为操作数常用在程序转移语句中，如

```
ORG      $8000
```

```
        LDY     #00
        TYA
START   STA     $200,Y
        INY
        BNE     START
        RTS
```

这是一个将内存区 $200～$2FF(2 页)清零的子程序,第 6 行语句中的操作数就是标号,它表示 Y 寄存器中的数值不为 0 时,转标号行继续对 2 页单元置 0。

2. 常数形式的操作数

常数一般系指十进制、十六进制数据,如以上示例中"STA $77"中的"$77"、"LDY #00"中的"00"都是常数,前一个是地址常数、后一个是立即数。

3. 表达式形式的常数

表达式可以由"+、-、*、/"符号连接而成,如:

LDA ABCD-1

表示把(ABCD-1)的值送入累加器。

六、注释

注释段是一个可选字段,注释内容必须以分号开始,注释的目的是为了使程序阅读方便。注释内容不属于程序本身,它的多少不影响程序的编译。

七、伪指令

伪指令是一类面向汇编程序的命令,它仅在程序的编译中执行一些特殊的操作。例如它在源程序汇编时,为目标程序分配一定的存储区域;给符号赋值;将指定的数据放入指定的

存储单元;留出存储数据使用的存储区域等等。

伪指令构成的语句也可以分为标号、操作码、操作数、注释四个字段,下面仅列出几个常用的伪指令。

1. 定义程序起始地址的伪指令 ORG

"ORG"是"ORiGin"(起点)的缩写,它的功能是在源程序汇编时,规定目标程序或数据存放的起始地址。"ORG "语句行中操作数字段的内容是存储单元的地址首址。例如:

```
        ORG     $8800
START   LDA     #$FF
        STA     $F00
        ……
```

由于使用了"ORG"伪指令,上述语句的目标程序将从地址"$8800"开始存放,同时标号"START"也与"$8800"等量。

若在上述程序后加入一句

ORG *+50

则在当前程序地址的基础上空出 50H 个字节,使后续程序从此地址开始向后存放。

2. 定义符号的伪指令 EQU

"EQU"是"EQUate"(等值)的缩写。"EQU "也称"等值"指令,其功能是用来给标号指定一个值。"EQU "语句必须带有标号,而且该标号也不允许再作它用。例如:

```
LAST    EQU  $C000
        ……
        JSR LAST
        ……
```

上述程序经过编译后,"LAST"标号将被赋值为"$C000 "这个地址,而"JSR LAST"也将被编译为"JSR $C000"。

注意在编写程序时，“EQU”语句必须放在程序的开始位置。

3. 定义字节的伪指令 DFB 或 DB

“DFB”是“DeFine Byte”(定义字节)的缩写。“DFB ”语句的功能是把指定字节或字节串存入从标号开始的连续一段单元中。例如：

```
        ORG    $9000
MEMB    DFB    $00,$11,$22,$33,$44,$55,'A,'B,'C
```

伪指令“ORG”规定了标号“MEMB”的起始地址是$9000，伪指令“DFB”指定字节串 00、11、22、33、44、55 中的内容以及字符 A、B、C 的 ASCII 码共 9 个数据，应该连续存放在标号“MEMB ”开始(即从$9000 开始)的连续单元中。其中第七、八、九三个地址单元存放的是字母 A、B、C 的 ASCII 码。

4. 定义字的伪指令 DW

“DW”是“Define Word”(定义字)的缩写。此指令定义一个 16 位的二进制数(即两个字节长度的数)，其低 8 位在第一字节，高 8 位在第二字节。例如：

```
WORD      DW  $FD11
```

此时，在“WORD”单元将存入 11H，“WORD+1”单元将存入 FDH。

伪指令还有一些，常用的就介绍这四个。

八、本书讨论中采用的汇编程序格式

目前，电脑学习机中还没有配备汇编语言工作系统，编写的汇编程序只能在 PC、286 以上的微机上进行编译，生成目标程序后，才能拿到学习机上运行。考虑到多数用户不具备微机条件，因此本书讨论中将不采用标准汇编语言程序格式，而

采用中华学习机机小汇编的程序格式。即：一般不使用伪指令和标号，所列程序以“地址、机器码、助记符、注释”的格式给出，对于程序的功能一般在程序开始位置单起一行说明。例如：

```
        $77、$78 单元赋值子程序
8000 A9 FF LDA  # $FF    ;取立即数于累加器
8002 85  77 STA  $77     ;$77、$78 两单元置入 FFH
8004 85  78 STA  $78
8006 60     RTS          ;子程序返回
```

这样列出程序清单，助记符和注释部分便于读者对程序的阅读、理解，比较适合初学者接受和掌握。

对于具备微机条件的读者，只要掌握了汇编语言的编程原理，把本书列出的小汇编程序，按规定套上汇编语言的标号和伪指令格式，就可得到标准的汇编语言程序。

第六节　电脑学习机硬件系统简介

电脑学习机的硬件系统，一般由主机、学习卡两部分组成，主机提供学习机的操作环境，学习卡提供学习机的软件工作系统，这是目前学习机的主流结构。但不论采取那种形式，硬件系统的组成是相同的，本节仅以流行结构为例讨论。

一、主机

电脑学习机的主机一般由主机芯片 6561（早期的产品为 9 片机，共包括：6527CPU、6528PPU、两片 6116、74LS139、74LS373、CD4069 和两片 74LS368。现在的产品已把这 9 块芯片集成封装在一起）、键盘、键盘接口、操纵器接口、录音机

接口、PAL 制式射频调制器、电源稳压器、打印机接口和 60 针插槽组成，较高档的产品还包含软盘驱动器和软驱接口等（见图 1－10）。有关各部分电路的结构、功能、工作原理等将在丛书的第十册《硬件与维修》中详细讨论。

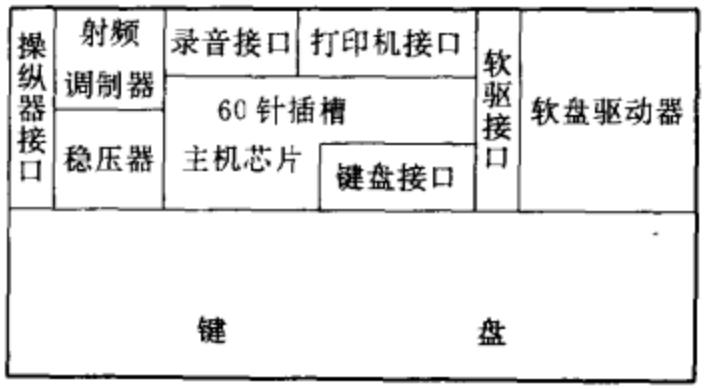


图 1－10　电脑学习机主机结构示意图

二、学习卡

学习卡一般由一块系统 ROM 芯片、两块 RAM 芯片和切换门电路组成，少数机型还留有扩展插槽和电池插座，用于扩充系统功能和断电保持记录之用，结构如图 1－11 所示。

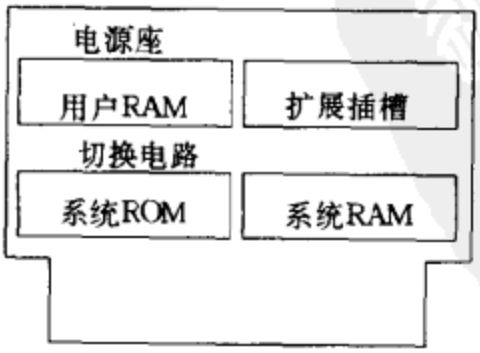


图1－11 学习卡结构示意图

系统 ROM 的容量一般在 512K～1024K 字节（4～8 兆位），其中有 256K 字节固化汉字一、二级字库和符号库，剩余

字节用于存放各系统软件，如 F BASIC 系统、浮点 BASIC 系统、中文处理系统、LOGO 语言系统以及各种应用软件。

系统 RAM 仅有 8K 字节，用于存放显示字模；

用户 RAM 一般为 8K 字节，高档机型为 32K 字节，用于存放用户编写的程序或编辑的文本文件等。

扩展插槽一般可插 128～256K 字节的 ROM 芯片，用于扩充系统功能，如挂接其它工作系统、各种应用软件、任天堂游戏软件等。

三、内部寄存器

6527 内部有六个可供编程使用的寄存器（A、X、Y、S、P、PC），其中除程序计数器 PC 是十六位之外，其它寄存器均为八位。

各寄存器的性能、作用如下：

1. 累加寄存器 A（Accumulator）

累加寄存器是一个八位寄存器（可存储 8 位二进制数据），用大写字母“A”表示，一般称为累加器或 A 寄存器。它同算术逻辑运算单元（ALU）共同完成各种算术逻辑运算。在通常的工作中，它既为 ALU 提供原始操作数据又担任存放 ALU 运算结果的任务。因此，在各寄存器中它是最活跃、使用最频繁的寄存器。

与累加器 A 有关的指令有：

实现内存与寄存器之间数据传送的指令 LDA、STA；实现寄存器与寄存器之间数据传送的指令 TAX、TXA、TAY、TYA；

进行算术、逻辑运算的指令 ADC、SBC、AND、ORA、EOR；

进行比较运算的指令 CMP；

进行堆栈操作的指令 PHA、PLA 等。

2. 变址寄存器 X、Y(Index Register)

变址寄存器共有两个,均为八位寄存器,分别用大写字母 X、Y 表示。它们主要用在变址寻址方式中存放地址偏移量(变址器),也常被当做计数器使用。它们可以通过指令控制置成一个常数,也可以方便地进行加 1、减 1、比较等简单操作,还可作为通用寄存器用于暂存数据。

与变址寄存器有关的指令有:

从内存单元取数至变址寄存器的指令 LDX、LDY;

将变址寄存器中的数据送入内存的指令 STX、STY;

累加器与变址寄存器相互传送数据的指令 TAX、TAY、TXA、TYA;

堆栈寄存器与变址寄存器之间传送数据的指令 TSX、TXS;

变址寄存器内部加 1、减 1 操作的指令 INX、INY、DEX、DEY;

变址寄存器比较运算的指令 CPX、CPY 等。

3. 堆栈指针寄存器 S(Stack Pointer)

它是专用于指示系统堆栈栈顶位置的八位寄存器。由于 6527 CPU 规定系统堆栈区设置在第一页($0100～$01FF),所以堆栈指针 S 有八位已能满足使用。工作中它指向堆栈地址的低八位,在有数据进栈时,S 内容自动加一;数据出栈时,S 内容自动减一。

与堆栈寄存器有关的指令有:

堆栈寄存器与变址寄存器之间传送数据的指令 TSX、TXS;

对累加器 A 进行压栈、出栈处理的指令 PHA、PLA;

对标志寄存器 P 进行压栈、出栈处理的指令 PHP、PLP。

4. 程序计数器 PC(Program Counter)

PC 是 6527CPU 中唯一的 16 位寄存器。它由两个 8 位寄存器组成，一个是高位地址程序计数器 PCH，用于表示地址的高位部分；一个是低位地址程序计数器 PCL，用于表示地址的低位部分。它主要用于存放下一条指令的首字节地址。当程序顺序执行时，每取出一个指令字节后，PC 自动加指令长度，为取下一个指令字节作好准备；当程序需要转移到新的地址时，PC 将放进要转移的目标地址。

5. 标志寄存器 P(Processor Status)

标志寄存器 P 也称状态寄存器，用大写字母 P 表示。它也是一个八位寄存器，工作中只使用了六位，D4、D5 位未用。这七个标志位的作用如下：

D7	D6	D5	D4	D3	D2	D1	D0
N	V			D	I	Z	C

(1)C——进位标志(Carry)

它标志着指令执行后最高位的进位状态。进行加法运算时，若最高位有进位则 C 位置 1、否则置 0；进行减法运算时，若最高位有借位则 C 位置 0、否则置 1。

(2)Z——零标志(Zero)

它表示本次操作后执行结果的情况。若结果为 0 则 Z 位置 1，否则置 0。

(3)I——中断禁止(屏蔽)标志(Interrupt Disable)

它表明是否允许中断。允许中断时 I 位为 0，禁止中断时 I 位为 1。由于 6527 CPU 的工作中一般仅使用非屏蔽中断，所以该位一般不用。

(4)D——十进制运算标志(Decimal Mode)

它表示 ALU 进行的运算是十进制运算还是二进制运算。十进制运算时D位置1,否则D位置0。D位的状态只对后续的加法或减法指令有作用。它只是在运算前对运算数据做出规定而不表示运算的结果。

(5)V——溢出标志(Overlow)

它表明本次运算结果的溢出状态。若有溢出则V位置1,否则V位置0。

(6)N——负数标志(Negative Flag)

它表示本次运算结果是否为负。它的实质是把本次运算结果的最高位(D7位)复制到N中。(对于带符号的八位二进制数,D7位为0是正数;D7位为1是负数。)

若V=0(无溢出时),N代表运算结果的符号:即运算结果为负数时N为1,否则N为0。

若V=1(有溢出时),则N代表的符号与运算结果的符号相反。即运算结果是负数时N=0,是正数时N=1。

对标志位的学习要注意掌握以下几点:

① 各指令对标志位的影响不尽相同(详见附录)。

② 寄存器P中各标志位的状态一般是指当前指令执行后的状态,所以标志位常用在执行条件转移时作为判断的依据。

③ 6527 CPU还可用专门指令对某些标志位进行操作。如"CLC"指令是对C位置0,对其余各位不影响;"CLD"指令是对D位置0,对其余各位不影响。

与标志寄存器有关的指令有:

进位标志设置、清除指令SEC、CLC;

中断标志设置、清除指令SEI、CLI;

十进制运算标志设置、清除指令SED、CLD;

溢出标志清除指令 CLV。

四、中断与复位

6527 CPU 的中断矢量为 $FFFA ～ $FFFB 和 $FFFE ～ $FFFF。其中 $FFFA、$FFFB 为非屏蔽中断 NMI 的地址，中断请求信号由 PPU 的第 19 脚发出，在场回扫的消隐期间触发，中断频率为 50Hz，中断间隔为 0.02 秒。$FFFE～ $FFFF 为可屏蔽中断 IRQ 的矢量地址，可屏蔽中断请求信号的发出由 CLI 指令控制，一般不使用。

6527 CPU 的复位地址存在 $FFFC 和 $FFFD 单元中。开机后，系统自动执行由 $FFFC、$FFFD 指定地址的复位程序。

五、6527 CPU 管理的内存分布

6527CPU 有 16 位地址线，最大寻址能力为 64K 字节。其内存地址分布情况如表 1－6。

表 1－6　6527CPU 管理的内存地址分布表

地址区间	用　途
$0000～ $00FF	系统 0 页
$0100～ $01FF	系统堆栈
$0200～ $02FF	卡通定义区
$0300～ $07FF	软件中间数据暂存区
$0800～ $1FFF	空区。影射于 $0000～ $07FF
$2000～ $3FFF	与 PPU 相连的 I/O 区
$4000～ $5FFF	CPU 的 I/O 区
$6000～ $7FFF	8K 空区。根据需要可扩充为 RAM 区可将其作为用户程序存储区。
$8000～ $FFFF	软件 ROM 区，共 32K 字节。软件容量大于 32K 时，在此区间有存储体切换

六、6528 PPU 管理的内存分布

6528 PPU 有 14 位地线，可管理 16K 内存，其管理的内存分布如表 1－7 所示。

表 1－7　6528 PPU 所管理的内存分布表

地 址 区	用　　途
$0000～$0FFF	图形库第 I 区，共 4K 空间，存储 256 个 8×8 点阵的卡通图形块。
$1000～$1FFF	图形库第 I 区，共 4K 空间，存储 256 个 8×8 点阵的背景图形、字符图形块。
$2000～$23BF	背景第一页(00 页)屏幕映射区，每一个单元存储一个 8×8 点阵的图形块代码。
$23C0～$23FF	背景第一页(00 页)配色区，每一个单元存储 16 个显示单元的配色代码。
$2400～$27BF	背景第二页(10 页)屏幕映射区。
$27C0～$27FF	背景第二页配色区。
$2800～$2BBF	背景第三页(01 页)屏幕映射区。
$2BC0～$2BFF	背景第三页配色区。
$2C00～$2FBF	背景第四页(11 页)屏幕映射区。
$2FC0～$2FFF	背景第四页配色区。
$3000～$3EFF	空区。映射于背景页。
$3F00～$3F0F	背景页配色代码定义区。其中 $3F00 为底背景配色区。
$3F10～$3F1F	卡通页配色代码定义区。
$3F20～$3FFF	空区。映射于 $3F00～$3F1F。

七、系统软开关

PPU 内部共有 8 个控制寄存器，通常称为系统软开关。地址为 $2000～$2007。CPU 正是通过对这些寄存器，控制

背景、卡通显示的。各软开关的功能如下：

1. $2000(只写寄存器)

D0 位：显示窗口的 X 坐标高位，0：为选背景 00 或 10 页；
1：为选背景 01 或 11 页。

D1 位：显示窗口的 Y 坐标高位 0：为选背景 00 或 01 页；
1：为选背景 10 或 11 页。

通过 D0、D1 位可以将显示窗口移到任意一页。

D2 位：PPU 地址读写方式。 0：地址加 1(横向)；
1：地址加 32(纵向)。

D3 位：卡通使用的字模首地址。0：从 $0000 开始；
1：从 $1000 开始。

D4 位：背景使用的字模首地址。0：从 $0000 开始；
1：从 $1000 开始。

D5 位：卡通大小控制。 0：8×8 点阵小卡通；
1：8×16 点阵大卡通。

D7 位：NMI 中断申请屏蔽。0：屏蔽；
1：允许。

2. $2001(只写寄存器)

D0 位：色彩禁止控制。0：有彩色输出；
1：无彩色输出。

D1 位：背景左一列显示控制。0：显示；1：不显示。

D2 位：卡通左一列显示控制。 …… …… …… ……

D3 位：背景全部显示控制。 …… …… …… ……

D4 位：卡通全部显示控制。 …… …… …… ……

D5 位：红色控制。 0：不着色；1：着色。

D6 位：绿色控制。 …… …… ……

D7 位：蓝色控制。 …… …… ……

3. $2002(只读寄存器)

D0－D4 位:未使用

D5 位:卡通 8 枚 OVER 标志。　0:不超过;1:超过。

D6 位:卡通冲突标志。　0:不冲突;

1:冲突。

D7 位:NMI 状态标记。　0:NMI 未到来;

1:到来。

注——通过读寄存器 $2002 可以得到 NMI 申请允许。

4. $2003(只写寄存器)

定义卡通工作页面首地址的低位(A0－A7)。

5. $2004(只写寄存器)

卡通面属性数据 D0－D7。

卡通面属性工作区在 PPU 内部,共 256 个字节。每一枚卡通的属性由 4 个字节组成,共可定义 64 枚卡通。(一般在 NMI 中断期间,数据从 CPU 向 PPU 作 DMA 传送。)当对 $2004 置数时,该数据可直接送至 PPU 的卡通属性工作区。

6. $2005(只写寄存器)

显示窗口设定,用来设置显示窗左上角的坐标。

第一次写 $2005 设定显示窗口的 X 坐标低位(X0～X7),第二次写 $2005 设定显示窗口的 Y 坐标低位(Y0～Y7),至于显示窗口的坐标高位(X8、Y8)将通过 $2000 的 D0、D1 位设定。

显示窗口的坐标范围:

X 方向:0～511 点(十六进制为 0～1FFH);

Y 方向:0～239、256～495 点(十六进制为 0～1EFH)。

7. $2006(只写寄存器)

PPU 地址寄存器,按高位、低位的顺序写两次,指定范围

为$0000～3FFF。

8. $2007(数据寄存器)

PPU 数据寄存器。当对$2007 读或写后,PPU 将根据$2000 单元 D2 位的状态,令 PPU 地址自动加 1 或加 32(20H)。

八、声音控制寄存器

6527 CPU 的声音控制寄存器地址为$4000～$4013 共 20 个单元,由它们组成五个发声声部,每个声部占用 4 个单元。其分配顺序如下:

- 第一主声部:$4000～$4003
- 第二主声部:$4004～$4007
- 第三主声部:$4008～$400B
- 第一副声部:$400C～$400F
- 第二副声部:$4010～$4013

各声部对应单元的功能相同。下面仅以第一主声部为例介绍各软开关的作用。

1. 音色音量控制寄存器$4000

$4000 是第一主声部的音色、音量控制软开关,各位功能如下:

D0～D3:为 15 级音量控制软开关。其值为 0 时关闭音量;为 1 时音量最小;为 15 时音量最大。

D4～D7:为音色控制软开关。其中

D4=1 时发声时间延长;

D5=1 时可发出各段衰减音;

D6=1 时将缩短发声时间。

2. 包络控制寄存器$4001

$4001 是第一主声部音形包络控制软开关。各位功能如下：

D0～D3：为 15 种包络形状控制开关，取值范围 0～15。

当取值等于 0 和等于 8 时无包络，发平直音；

当取值等于 1～7 时，发音频率从高到低变化；

当取值等于 9～5 时，发声频率则从低到高；

当取值等于 1、9 时发声时间最短；

取值等于 7、15 时发声时间最长。

D4～D6：为 8 级包络周期控制开关。调整它们的取值，可模拟声差极大的效果音响。它们与 $4000 的 D5 位配合可发出多级衰减音。

D7：为包络开关位。D7＝1 时包络有效。

3. 音调细调控制寄存器 $4002

$4002 是第一主声部音调细调控制软开关。使用它可以细调发声音调，取值范围 0～255，取值越大发声频率越低。

4. 音调粗调控制寄存器 $4003

$4003 是第一主声部音调粗调开关。使用它可以粗调发声音调，取值范围 0～7，取值越大频率越低。

5. 发声开关控制寄存器 $4015

$4015 是启动发声声部的控制寄存器，写入不同的数据，可选择不同的发声声部。从 D0～D4 位的控制顺序依次是第一、二、三主声部和第一、二副声部。

九、DMA 控制寄存器 $4014

DMA 是指存储器与外设之间的直接数据传送，在数据的传送过程中，不须经过 CPU 的中转，故数据传送速度最快，一般用于存储器与外设间需要进行大量而快速传送数据

的场合。

CPU 中的 $4014 寄存器就是 DMA 传送控制寄存器。它可以指定 PRAM 向 PPU 中卡通属性存储器 DMA 传送的首地址。例如：

```
LDA  # $00
STA  $2003
LDA  # $02
STA  $4014
```

在 CPU 响应 NMI 中断期间，将把 PRAM 中 $200～$2FF 中存放的卡通属性(共 64 组)用 DMA 方式传送到 PPU 属性寄存器中。

第二章　6527的寻址方式

什么是“寻址方式”呢？

简单地说，“寻址方式”就是寻找指令地址或操作数地址的方式。

我们知道，控制计算机各部分协调工作的命令通常称为指令，每条指令都规定和指明了计算机的某项操作内容。

一条指令一般可分为两部分：第一部分（主要部分）称为操作码，它指明该条指令的操作性质、具有何种功能、将进行什么样的操作；第二部分（从属部分）称为操作数，它指明了参与操作或运算的操作对象。操作对象一般分三类：第一类就是操作数据本身；第二类为操作数据所存放单元的地址；第三类为操作数据所在地址的地址。

何谓“地址”？

数据和指令在存储器中存放的位置就称为地址。

地址有两类：一类是指令地址（即指令的存放位置）；一类是操作数地址（即操作数据的存放位置）。

本章将重点讨论这两类地址的寻找方式。

第一节　6527指令系统的特点

各种型号的CPU都具有许多不同的指令，如取数、存数、相加、相减、比较、移位、转移……等等。这些指令的全体就组成了该型CPU的指令系统。显然，不同型号的CPU将

具有不同的指令系统。由于 6527 CPU 属于 65 系列，所以它的指令系统与中华学习机 6502 指令系统相同。它共有 13 种寻址方式、56 条基本指令，寻址方式和基本指令组合，可衍生 151 条操作指令，其主要特点如下：

1. 支持零页寻址

虽然 6527 CPU 的内部寄存器较少，但它的指令系统包含零页寻址方式，CPU 可直接访问 $00～$FF 共 256 个单元。由于零页寻址方式的指令字节少、占用机器周期短，所以可减少程序对内存的占用，缩短程序的运行时间。

2. 指令格式统一

6527 的指令长度为 1～3 个字节，有统一的指令格式。规定第一个字节必须是操作码，由它决定 6527 完成一种确定的运算或操作。操作码后面紧跟的是操作数或操作数地址，占用一个或两个字节，占用字节的多少视寻址方式而定。

由于指令格式统一，方便了 6527 CPU 开发系统软件的编写。

3. 外设访问简便

6527 CPU 与外设之间交换数据或信息采集，采用了存储器映像方式，凡是用于访问存储器的指令都可以用来访问外设，访问外设的指令多，访问外设的范围广，给使用带来了极大的方便。

4. 高效的显示方式

6527 CPU 对显示的控制仅通过 8 个系统软开关向 PPU 发布命令，CPU 和 PPU 并行工作，显示处理占用机时短，工作效率高。

5. 简单的发声控制

6527 CPU 内含 PSG（Programmable Sound Gener —

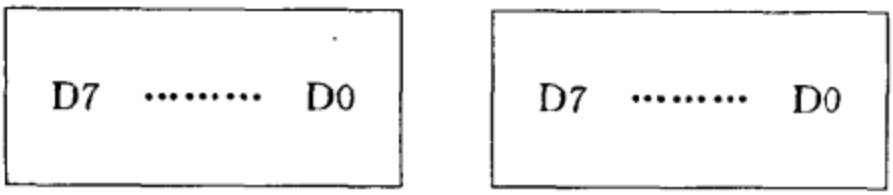
ator)——可编程音响发生器。旋律的演奏或效果音响的发出，仅对系统发声软开关操作即可实现，不要专门的发声指令进行控制，发声操作简单容易。

第二节　立即寻址

在指令的操作对象部分直接给出了操作数本身的寻址方式称为立即寻址方式(Imediate Addressing)，这时的操作数称为立即数。

一、立即寻址方式的指令形式

立即寻址方式的指令形式为：



操作码(OP)　　操作数(nn)

其汇编指令形式为：

助记符 #$nn

操作数部分的"#$"为立即数的符号，"nn"表示十六进制数据。

例如：指令代码

A9　7F

是一条两字节指令，用汇编语言写出就是

LDA　#$7F

其中"A9"是操作码，它表示取一个数据装入累加器 A 的操作。执行这个指令后，累加器 A 中的数值就是后面的立即数

7FH。也就是说，这条指令的功能就是将立即数 7FH 送入累加器 A，即 7FH→A。

二、说明

1. 立即寻址方式的指令，常用于设置程序所需要的各种固定初值、累加器中的内容与某一个立即数相加减、逻辑运算以及 A、X、Y 寄存器中的内容与立即数比较等。

2. 立即寻址方式的指令长度均为两个字节。

3. 立即寻址方式指令的操作数部分是操作数本身，不是地址。

4. 立即数为两位十六进制数，取值范围为 00～FFH。

三、立即寻址方式指令

立即寻址方式的指令共有 13 条。

1. 立即数送累加器指令(A9)

格式：A9 nn LDA # $nn

功用：取立即数送累加器 A。

注释：LDA 来源于“Load Accumulator”，是“装载累加器”的意思，在此表示把数据装入 A 寄存器。

说明：此指令执行后将改变标志寄存器的 N、Z 标志。

示例：

```
A9  00  LDA  # $00     ;累加器清零
A9  FF  LDA  # $FF     ;累加器置入 FFH
```

2. 立即数送 X 变址寄存器指令(A2)

格式：A2 nn LDX # $nn

功用：取立即数送变址寄存器 X。

注释：LDX 来源于“Load index register X”，是“装载变

址寄存器 X"的意思。

说明:该指令将改变标志寄存器的 N、Z 标志。

示例:

```
A2 00 LDX # $ 00 ;变址寄存器 X 清零
A2 88 LDX # $ 88 ;变址寄存器 X 置入 88H
```

3. 立即数送 Y 变址寄存器指令(A0)

格式:A0 nn LDY # $ nn

功用:取立即数送变址寄存器 Y。

注释:LDY 来源于"Load index register Y",是"装载变址寄存器 Y"的意思。

说明:该指令将改变标志寄存器的 N、Z 标志。

示例:

```
A0 00 LDY # $ 00 ;变址寄存器 Y 清零
A0 99 LDY # $ 99 ;变址寄存器 Y 置入 99H
```

4. 累加器与立即数带进位加运算指令(69)

格式:69 nn ADC # $ nn

功用:累加器 A 中的内容与立即数进行带进位加法运算,运算后的结果仍然存在累加器 A 中。

注释:ADC 来源于"ADd With Carry",是"带进位加"的意思。

说明:该指令执行后,将改变标志寄存器的 N、V、Z、C 标志。

示例 1:(假定进位 C=0)

```
A9   53   LDA   # $ 53      ;53→A
69   67   ADC   # $ 67      ;A+C+67→A
```

上述程序执行后,累加器 A 中的内容为 BAH。现在,因运算结果的 D7=1,所以标志寄存器的 N=1,C=0。

示例 2:(假定进位 C=1)

```
A9   53   LDA   # $53      ;53→A
69   67   ADC   # $67      ;A+C+67→A
```

上述程序执行后,累加器 A 中的内容为 BBH。N=1,C=0。

5. 累加器与立即数带进位减运算指令(E9)

格式:E9 nn SBC # $nn

功用:累加器 A 中的内容减立即数、减进位,运算后的结果仍然存在累加器 A 中。

注释:SBC 来源于"SuBtract With Carry",是"带进位减"的意思。

说明:该指令将改变标志寄存器的 N、V、Z、C 标志。

示例 1:(假定进位 C=1)

```
A9   77   LDA   # $77
E9   48   SBC   # $48      ;A−C̄→A
```

上述程序执行后,累加器中的内容为 2FH

示例 2:(假定进位 C=0)

```
A9   77   LDA   # $77
E9   48   SBC   # $48      ;A−C̄→A
```

上述程序执行后,累加器中的内容为 2EH

6. 累加器与立即数"与"逻辑运算指令(29)

格式:29 nn AND # $nn

功用:累加器 A 中的内容与立即数进行按位"与"逻辑运算,运算结果仍然存在累加器 A 中。运算示意如图 2−1 所示。

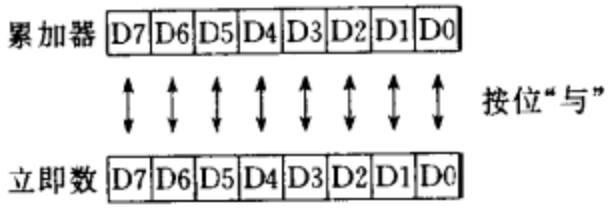


图 2－1 “与”运算示意图

注释：AND 在英文中为“与”的意思。

说明：该指令将改变标志寄存器的 N、Z 标志。

示例：

```
A9  38  LDA  # $ 38     ;累加器 A 置入 38
29  77  AND  # $ 77     ;38 AND 77
```

运算示意如图 2－2 所示。

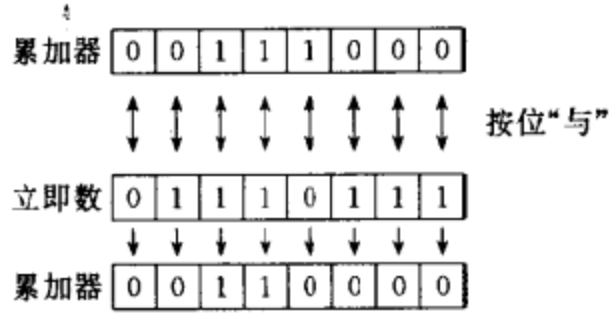


图 2－2 “与”运算示意图

可见，上述程序执行后，累加器 A 中的数据为 30H。

7. 累加器与立即数“或”逻辑运算指令(09)

格式：09 nh ORA # $ nn

功用：累加器 A 中的内容与立即数进行或逻辑运算，运算结果仍然存在累加器 A 中。

注释：ORA 来源于“OR With Accumulator”，是累加器的内容与立即数进行“或”逻辑运算的意思。

说明：该指令将改变标志寄存器的 N、Z 标志。

示例：

```
A9  37  LDA  # $37      ;累加器 A 等于 37
09  78  ORA  # $78      ;37 OR 78
```

运算示意如图 2—3 所示。

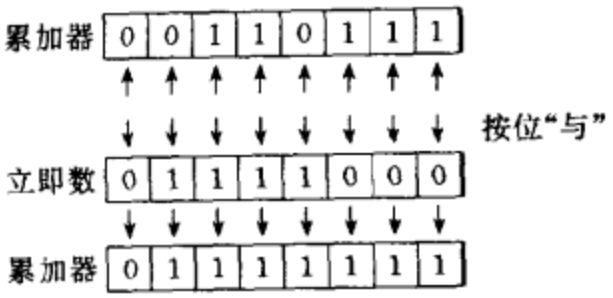


图 2—3 “或”运算示意图

可见，上述程序执行后，累加器 A 中的数据为 7FH。

8. 累加器与立即数“异或”逻辑运算指令(49)

格式：49 nn EOR # $nn

功用：累加器 A 中的内容与立即数进行异或逻辑运算，运算结果仍然存在累加器 A 中。

注释：EOR 来源于“Exclusive OR”，是累加器的内容与立即数进行“异或”逻辑运算的意思。

说明：该指令将改变标志寄存器的 N、Z 标志。

示例：

```
A9 30 LDA  # $30      ;累加器 A 中的内容等于 30
49 FF EOR  # $FF      ;30 EOR FF
```

运算示意如图 2—4 所示。

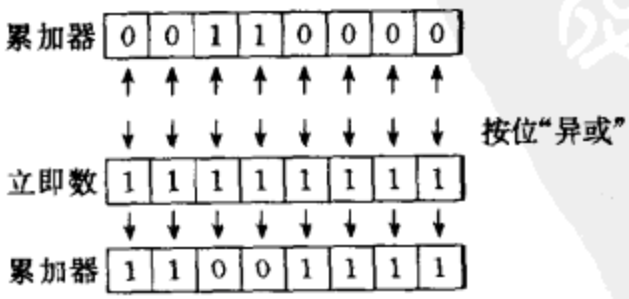


图 2—4 “异或”运算示意图

可见，上述程序执行后，累加器 A 中的数据为 CFH。

9. 累加器与立即数比较指令(C9)

格式:C9 nn CMP # $nn

功用:累加器 A 中的内容与立即数比较,比较后累加器 A 中的内容不变,仅改变标志寄存器的 N、Z、C 标志。

注释:CMP 来源于"CoMPare",是"比较"的意思。

说明:比较指令的操作过程是:拿累加器 A 中的数据带进位减所比较的立即数,然后判断结果情况。

若结果为 0 则置 Z=1,否则 Z=0;

若有借位则 C=0,否则 C=1;

若结果的 D7=1 则 N=1,否则 N=0。

示例:

```
A9 44 LDA # $44 ;累加器置入 44H
C9 30 CMP # $30 ;累加器 A 中的内容与 30H 比较
```

执行上述程序后,因

44−30=14

结果不等于 0、没有借位、结果的 D7=0,所以标志寄存器的 Z=0、C=1、N=0。

10. X 变址寄存器与立即数比较指令(E0)

格式:E0 nn CPX # $nn

功用:X 变址寄存器中的内容与立即数进行比较,X 中的内容不变,仅改变标志寄存器的 N、Z、C 标志。

注释:CPX 来源于"ComPare With index register X",是"与 X 变址寄存器内容比较"的意思。

说明:比较操作同(C9)。

示例:

```
E0 30 CPX # $30 ;X 变址寄存器的内容与 30H 比较
```

11. Y 变址寄存器与立即数比较指令(C0)

格式:C0 nn CPY # $nn

功用:Y 变址寄存器中的内容与立即数进行比较，Y 中的内容不变,仅改变标志寄存器的 N、Z、C 标志。

注释:CPY 来源于“ComPare With index register Y”,是“与 Y 变址寄存器内容比较”的意思。

说明:比较操作同(C9)

示例:

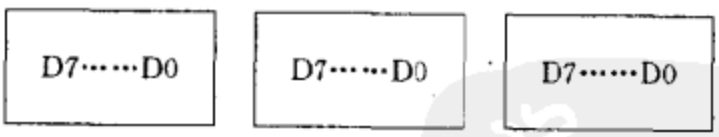
```
C0 30 CPY # $30 ;Y 变址寄存器的内容与 30H 比较
```

第三节 绝对寻址

操作数部分的内容，直接给出操作数在内存中实际地址的寻址方式,称为绝对寻址(Absolute Addressing)。

一、绝对寻址方式的指令形式

绝对寻址方式的指令形式为



操作码(OP)　操作数(低位)n1　操作数(高位)n2

其汇编指令格式为:

助记符 $n2n1

其中“$”符号表示其后面的操作数为内存单元的地址、“n2n1”为 4 位十六进制数据,“n2”为十六进制地址高位、“n1”为十六进制地址低位。如绝对寻址方式的累加器取数指令

LDA $8000

表示把内存地址为 $8000 单元的内容送入累加器 A，即

(＄8000)→A

这里的“＄8000”表示绝对地址，“(＄8000)”表示＄8000 单元所存储的内容。

要注意

LDA ＃＄7F

LDA ＄8000

指令的本质区别。虽然它们的助记符都是“LDA”，都是向累加器 A 中置数，但它们的操作码不同(前者 A9、后者 AD)，表示寻址方式不同，前者是立即数寻址、后者是绝对寻址。

二、绝对寻址方式说明

1. 绝对寻址方式指令的长度都是三个字节。

2. 绝对寻址方式使用两个字节存放操作数的地址，取值范围为 0000～FFFFH，故可以访问 64K 存储器中的任何一个地址。

3. 绝对寻址方式指令中地址的表示顺序如下：在机器码指令中，地址的低位在前、高位在后；但在汇编指令中地址的高位在前、低位在后。如“LDA ＄8000”指令的机器码指令为

AD 00 80

三、绝对寻址方式的指令

绝对寻址方式的指令共有 23 条。

1. 从内存单元取数送入寄存器指令

格式：AC n1 n2 LDY ＄n2n1

AD n1 n2 LDA ＄n2n1

AE n1 n2 LDX ＄n2n1

功用：从内存地址单元中取数送入寄存器。

说明：以上三条指令将改变标志寄存器的N、Z位。

示例：

```
AD 00 80 LDA  $8000 ; $8000)→A
AE 01 80 LDX  $8001 ; $8001)→X
AC 02 80 LDY  $8002 ; $8002)→Y
```

2. 寄存器中的数据送入内存单元指令

格式：8C n1 n2 STY $n2n1

8D n1 n2 STA $n2n1

8E n1 n2 STX $n2n1

功用：把寄存器中的数据送入指定地址的内存单元。

注释：ST来源于“STore”，是“存储”的意思。“STY”是把Y变址寄存器的内容置入指定的内存地址单元的意思。

说明：以上三条指令不影响标志寄存器。

示例：

```
A9 55 LDA  # $55
A2 56 LDX  # $56
A0 57 LDY  # $57
8D 10 80 STA  $8010 ; $8010=55
8E 11 80 STX  $8011 ; $8011=56
8C 12 80 STY  $8012 ; $8012=57
```

3. 无条件转移指令(4C)

格式：4C n1 n2 JMP $n2n1

功用：结束当前程序的操作，无条件地转移到指定地址单元，继续执行。

注释：JMP来源于“JuMP”，是“跳”的意思，在此表示跳转到指定地址，继续执行程序。

说明：该指令不影响标志位。

示例：

4C 00 90 JMP $9000 ；跳转到地址为$9000的单元继续执行程序。

4. 转子指令(20)

格式：20 n1 n2 JSR $n2n1

功用：保存当前指令最后一个字节的地址，调用指定地址的子程序。

注释：JSR来源于“Jump to SubRoutine”，是“跳到子程序”的意思，在此表示调用子程序。

说明：该指令不影响标志位。

示例：

20 00 90 JSR $9000 ；调用地址为$9000的子程序

5. 存储器内容加1指令(EE)

格式：EE n1 n2 INC $n2n1

功用：令指定地址的内存单元的内容加1。

注释：INC来源于“INCrement”，是“递增1”的意思。

说明：该指令将改变标志寄存器的N、Z标志。

示例：

```
A9  55      LDA  #$55     ;55H→A
8D  00  80  STA  $8000    ;$8000=55
EE  00  80  INC  $8000    ;$8000=56
```

6. 存储器内存减1指令(CE)

格式：CE n1 n2 DEC $n2n1

功用：令指定地址的内存单元的内容减1。

注释：DEC来源于“DECrement”，是“递减1”的意思。

说明：该指令将改变标志寄存器的N、Z标志。

示例：

```
A9  55      LDA  # $ 55     ;55H→A
8D  00  80  STA  $ 8000     ; $ 8000=55
CE  00  80  DEC  $ 8000     ; $ 8000=54
```

7. 累加器与存储器带进位加指令(6D)

格式：6D n1 n2 ADC $n2n1

功用：令累加器的内容与指定地址的内存单元的内容相加，同时加进位，结果仍存在累加器 A 中。

说明：该指令将改变标志寄存器的 N、V、Z、C 标志。

示例：(假定进位标志 C=0)

```
A9  55      LDA  # $ 55     ;55H→A
8D  00  80  STA  $ 8000     ; $ 8000=55
A9  44      LDA  # $ 44     ;44H→A
6D  00  80  ADC  $ 8000     ;A+C+ $ 8000
                            ;=44+0+55=99
8D  01  80  STA  $ 8001     ; $ 8001=99
```

8. 累加器与存储器带进位减指令(ED)

格式：ED n1 n2 SBC $n2n1

功用：令累加器的内容与指定地址的内存单元的内容相减，同时减进位的非，结果仍存在累加器 A 中。

说明：该指令将改变标志寄存器的 N、V、Z、C 标志。

示例 1：(假定进位标志 C=1)

```
A9  55      LDA  # $ 55     ;55H→A
8D  00  80  STA  $ 8000     ; $ 8000=55
A9  76      LDA  # $ 76     ;76H→A
ED  00  80  SBC  $ 8000     ;A−C̄− $ 8000
                            ;=76−0−55=21
```

```
8D  01  80  STA  $8001     ;$8001=21
```

示例2:(假定进位标志C=0)

```
A9  55      LDA  #$55     ;55H→A
8D  00  80  STA  $8000     ;$8000=55
A9  76      LDA  #$76     ;76H→A
ED  00  80  SBC  $8000     ;A-C̄-$8000
                           ;=76-1-55=20
8D  01  80  STA  $8001     ;$8001=20
```

9. 逻辑运算指令

格式:2D n1 n2 AND $n2n1

4D n1 n2 EOR $n2n1

0D n1 n2 ORA $n2n1

功用:累加器A中的数据与指定内存单元的数据进行逻辑运算,运算后的结果仍然存在累加器中。

说明:以上三条指令执行后,将改变标志寄存器的N、Z标志。

示例:

```
A9  58      LDA  #$58     ;58H→A
8D  00  A0  STA  $A000    ;$A000=58
A9  43      LDA  #$43     ;43H→A
2D  00  A0  AND  $A000    ;43 AND 58=40
8D  01  A0  STA  $A001    ;$A001=40
A9  55      LDA  #$55     ;55H→A
4D  01  A0  EOR  $A001    ;55 EOR 40=15
8D  02  A0  STA  $A002    ;$A002=15
A9  12      LDA  #$12     ;12H→A
0D  02  A0  ORA  $A002    ;12 OR 15=17
```

```
8D  03  A0  STA  $A003     ;$A003=17
```

示例程序执行后，

$A000=58

$A001=40

$A002=15

$A003=17

10. 存储器位测试指令(2C)

格式：2C n1 n2 BIT $n2n1

功用：取内存单元内容的最高位(D7 位)送标志寄存器的 N 位；次高位(D6 位)送标志寄存器的 V 位；取累加器 A 中的内容与内存单元的内容进行逻辑与运算，若结果为零置标志寄存器的 Z=1，若不为零则置 Z=0。

位测试后，累加器 A 和内存单元中的内容均不改变。

注释：BIT 在英文中为“位”的意思，在此表示对指定单元的内容进行按位测试。

示例：

```
A9  86      LDA  #$86
8D  00  80  STA  $8000    ;$8000=86
A9  7F      LDA  #$7F     ;7FH→A
2C  00  80  BIT  $8000    ;86 AND 7F
                          ;N=1
                          ;V=0
                          ;Z=0
```

11. 算术左移位指令(0E)

格式：0 n1 n2 ASL $n2n1

功用：指定内存单元内容的各位均向左移一位。其中原 D7 位移入进位 C、原 D0 位填入 0。移位示意如图 2－5 所示。

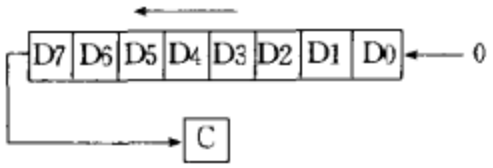


图 2—5　算术左移位示意图

注释：ASL 来源于“Arithmetic Shift Left”，是“算术左移”的意思。

说明：该指令将改变标志寄存器的 N、Z、C 标志。

示例：

```
A9  77      LDA  # $77
8D  00  80  STA  $8000     ; $8000=77
0E  00  80  ASL  $8000     ; $8000=EE
AD  00  80  LDA  $8000
8D  01  80  STA  $8001
0E  00  80  ASL  $8000     ; $8000=DC
AD  00  80  LDA  $8000
8D  02  80  STA  $8002
0E  00  80  ASL  $8000     ; $8000=B8
AD  00  80  LDA  $8000
8D  03  80  STA  $8003
```

执行上述程序后

$8000=B8

$8001=EE

$8002=DC

$8003=B8

算术左移位指令在编程中常用于乘 2 运算。

12. 逻辑右移位指令(4E)

格式：4E n1 n2 LSR $n2n1

功用：指定内存单元内容的各位全部向右移一位。原 D0

位移入进位 C、原 D7 位填入 0。移位情况如图 2－6 所示。

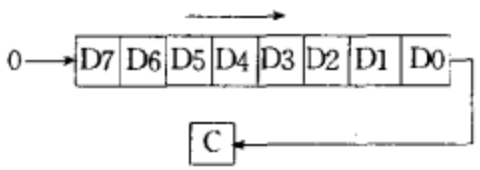


图 2－6　算术右移位示意图

注释：LSR 来源于“Logical Shift Rihgt”，是“逻辑右移”的意思。

说明：该指令将改变标志寄存器的 Z、C 标志，并令 N 标志位等于 0。

示例：

```
A9  77      LDA  # $ 77
8D  00  80  STA  $ 8000     ; $ 8000=77
4E  00  80  LSR  $ 8000     ; $ 8000=3B
AD  00  80  LDA  $ 8000
8D  01  80  STA  $ 8001
4E  00  80  LSR  $ 8000     ; $ 8000=1D
AD  00  80  LDA  $ 8000
8D  02  80  STA  $ 8002
4E  00  80  LSR  $ 8000     ; $ 8000=0E
AD  00  80  LDA  $ 8000
8D  03  80  STA  $ 8003
```

执行上述程序后

$ 8000＝0E

$ 8001＝3B

$ 8002＝1D

$ 8003＝0E

右移位指令在编程中常用于无符号数的除 2 运算。

13. 循环左移位指令(2E)

格式:2E n1 n2 ROL $n2n1

功用:指定内存单元内容的各位循环左移一位,原D7位移入进位C、原进位C移入D0位。移位示意如图2-7所示。

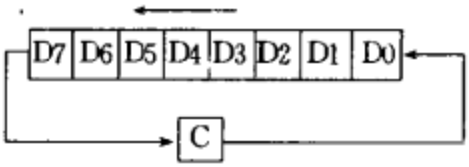


图2-7 循环左移位示意图

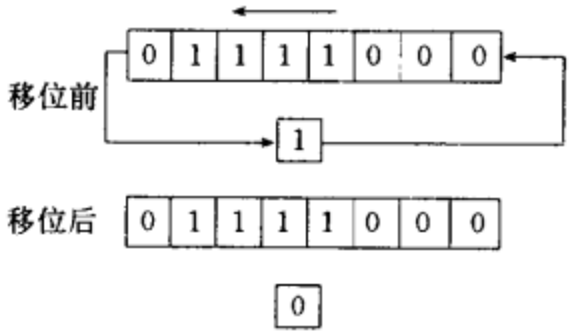
注释:ROL来源于"ROtate Left",是"向左循环"的意思。在此表示循环左移位。

说明:该指令将影响标志寄存器的N、Z、C位。与"左移位"相对应,为方便称呼,循环左移位常称为"左转位"。

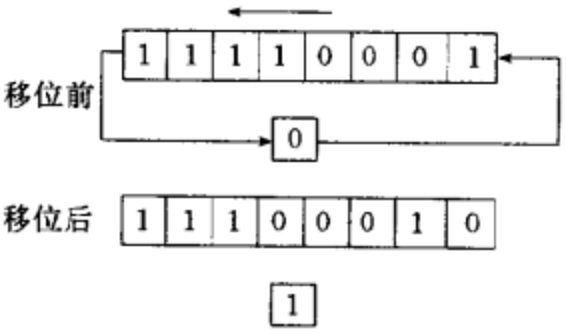
示例:(假定移位前,进位C=1)

```
A9  78      LDA  # $78
8D  00  C0  STA  $C000  ;$C000=78
2E  00  C0  ROL  $C000  ;$C000 循环左移一位
AD  00  C0  LDA  $C000
8D  00  80  STA  $8000  ;$C000→$8000
2E  00  C0  ROL  $C000  ;$C000 循环左移一位
AD  00  C0  LDA  $C000
8D  01  80  STA  $8001  ;$C000→$8001
2E  00  C0  ROL  $C000  ;$C000 循环左移一位
AD  00  C0  LDA  $C000
8D  01  80  STA  $8002  ;$C000→$8002
```

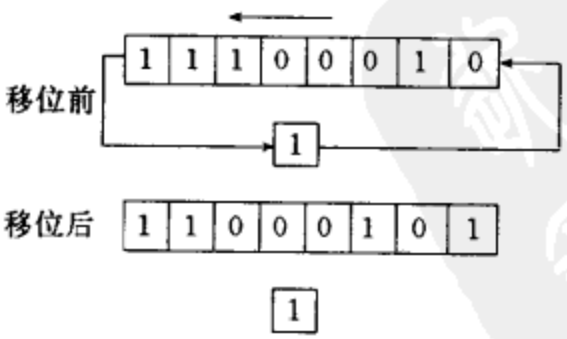
上例程序的执行中,$C000单元的三次循环左移操作如图2-8所示。



（A） 第一次循环左移



（B） 第二次循环左移



（C） 第三次循环左移

图 2—8 循环左移示意图

可见，程序执行结束后，

$8000=F1

$8001=E2

$8002=C5

$C000=C5

14. 循环右移位指令(6E)

格式：6E n1 n2 ROR $n2n1

功用：指定内存单元内容的各位循环右移一位，原D0位移入进位C、原进位C移入D7位。移位示意如图2－9所示。

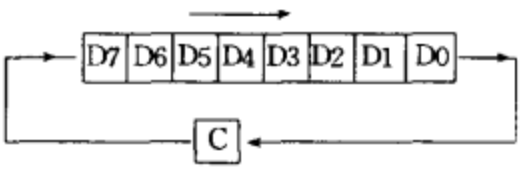


图2－9　循环右移位示意图

注释：ROR来源于“ROtate Right”，是“向右循环”的意思。在此表示循环右移位。

说明：该指令将影响标志寄存器的N、Z、C位。循环右移位也可称为“右转位”。

示例：(假定移位前，进位C=1)

```
A9  78      LDA  #$78
8D  00  C0  STA  $C000     ;$C000=78
6E  00  C0  ROR  $C000     ;$C000 循环右移一位
```

上例程序的执行中，$C000单元的一次循环右移操作如图2－10所示。

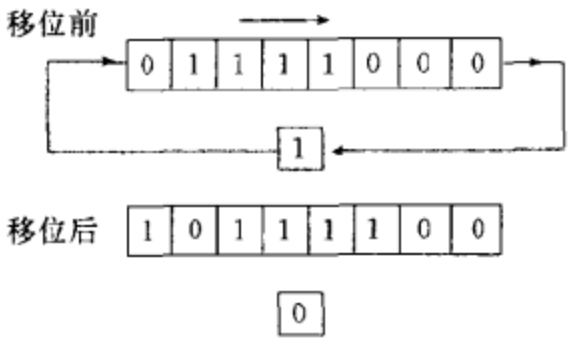


图 2—10　循环右移一次示意图

循环右移位一次后，$C000 单元的内容为 BCH。

15. 寄存器与存储器比较指令

格式：CC n1 n2 CPY $n2n1

CD n1 n2 CMP $n2n1

EC n1 n2 CPX $n2n1

功用：把寄存器中的数据与指定地址的内存单元中的数据比较。比较后，寄存器、存储器中的数据不变，仅改变标志寄存器的 N、Z、C 位。

说明：以上三条指令与立即寻址比较指令相比，除比较的数据是存储器中的数据之外，比较操作与立即寻址相同，请参阅第二节。

第四节　零 页 寻 址

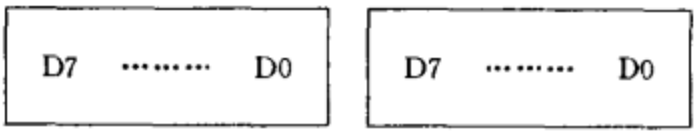
为讨论中叙述方便，一般把 CPU 访问的 64K 存储器空间以 256 个字节为单位划分为若干段，每段称为一页（在此形象地把全部存储器看成一本字数为 64K 的书，每 256 个字为一页）。页号从 0 开始，依次为 0 页、1 页、2 页……。实际上，存储器的页号就是存储器的高位地址码。零页存储器的

地址范围就是$0000～$00FF，对零页单元来说，由于高位地址恒等于零，故零页单元的存储器地址一般由一个字节表示。

对于零页的地址查询称为“零页寻址”(Zero－page Addressing)。零页寻址方式表达的操作数，实际上就是操作数的地址，该地址范围就限制在$00～$FF。

一、零页寻址方式的指令形式

零页寻址方式的指令形式为：



操作码(OP)　　操作数(地址)

其汇编指令形式为：

助记符 $nn

操作数部分的“$”为存储器地址的符号，“nn”表示十六进制数据。

例如：指令代码

A5 7F

是一条两字节指令，用汇编语言写出就是

LDA $7F

其中“A5”是操作码，它表示将进行从某个零页地址单元中取出数据装入累加器A的操作。执行这个指令后，累加器A中的数值就是后面的零页地址单元中的数据。也就是说，这条指令的功能就是把$7F中的数据送入累加器A，即($7F)→A。

注意——

A5 7F　　LDA $7F

AD 7F 00 LDA $007F

两条指令的功能是相同的，都是把$007F单元的数据送累加器A。但前者是零页寻址、后者是绝对寻址，前者比后者少用一个字节。

零页寻址和绝对寻址的操作对象部分都是操作数所在的地址，不同的是前者只能在零页范围的256个字节内寻址，而后者可以在64K存储单元中的任何一个单元寻址。

二、零页寻址方式说明

1. 零页寻址方式均为两字节指令，写成机器码时，第一个字节是操作码，第二个字节为操作数的零页地址。

2. 由于零页中操作数的地址仅限于零页范围（$0000～$00FF），故在表示操作数的地址码时，不必给出地址码的高位（高位地址约定为00），因而指令长度可以减少一个字节。

3. 零页寻址方式中的操作数是“$nn”，“nn”代表两位十六进制数。

4. 零页寻址指令占用字节少，执行速度快，在节省内存和缩短运行时间上，具有无比优越性。

三、零页寻址方式指令

零页寻址方式指令共有20条。

1. 存储器取数至寄存器指令

格式：A4 nn LDY $nn

A5 nn LDA $nn

A6 nn LDX $nn

功用：取出指定零页存储单元的数据置入寄存器。

说明:以上三条指令将影响标志寄存器的 N、Z 位。

LDY $nn 与 LDY $00nn

LDA $nn 与 LDA $00nn

LDX $nn 与 LDX $00nn

的功能是相同的,请参阅绝对寻址指令的介绍。

2. 寄存器的数据送入存储器

格式:84 nn STY $nn

85 nn STA $nn

86 nn STX $nn

功用:把寄存器中的数据存入指定的零页存储单元中

说明:以上三条指令均不影响标志寄存器。

STY $nn 与 STY $00nn

STA $nn 与 STA $00nn

STX $nn 与 STX $00nn

的功能是相同的,请参阅绝对寻址指令的介绍。

3. 加法运算指令(65)

格式:65 nn ADC $nn

功用:累加器 A、指定的零页单元内容、进位 C 三数相加,结果仍存在累加器 A 中。

说明:ADC $nn 与 ADC $00nn 的功能相同,请参阅绝对寻址指令的介绍。

4. 减法运算指令(E5)

格式:E5 nn SBC $nn

功用:累加器 A 中的数据减去指定零页单元的数据和进位的非,结果仍存在累加器 A 中。

说明:SBC $nn 与 SBC $00nn 的功能相同,请参阅绝对寻址指令的介绍。

5. 加 1 指令(E6)

格式:E6 nn INC $nn

功用:令指定零页单元中的内容加 1。

说明:INC $nn 与 INC $00nn 的功能相同,请参阅绝对寻址指令的介绍。

6. 减 1 指令(C6)

格式:C6 nn DEC $nn

功用:令指定零页单元中的内容减 1。

说明:DEC $nn 与 DEC $00nn 的功能相同,请参阅绝对寻址指令的介绍。

7. 逻辑运算指令

格式:05 nn ORA $nn

25 nn AND $nn

45 nn EOR $nn

功用:累加器 A 中的内容与指定零页单元的数据进行逻辑运算,运算后结果仍然存在累加器 A 中。

说明:ORA $nn 与 ORA $00nn

AND $nn 与 AND $00nn

EOR $nn 与 EOR $00nn

的功能是相同的,请参阅绝对寻址指令的介绍。

8. 位测试指令(24)

格式:24 nn BIT $nn

功用:测试指定零页单元中的内容各位。

说明:BIT $nn 与 BIT $00nn 的功能相同,请参阅绝对寻址指令的介绍。

9. 左移位指令(06)

格式:06 nn ASL $nn

功用：令指定零页单元中的内容左移一位。

说明：ASL $nn 与 ASL $00nn 的功能相同，请参阅绝对寻址指令的介绍。

10. 右移位指令(46)

格式：46 nn LSR $nn

功用：令指定零页单元中的内容右移一位。

说明：LSR $nn 与 LSR $00nn 的功能相同，请参阅绝对寻址指令的介绍。

11. 循环左移位指令(26)

格式：26 nn ROL $nn

功用：令指定零页单元中的内容循环左移一位。

说明：ROL $nn 与 ROL $00nn 的功能相同，请参阅绝对寻址指令的介绍。

12. 循环右移位指令(66)

格式：66 nn ROR $nn

功用：令指定零页单元中的内容循环右移一位。

说明：ROR $nn 与 ROR $00nn 的功能相同，请参阅绝对寻址指令的介绍。

13. 比较指令

格式：C4 nn CPY $nn

C5 nn CMP $nn

E4 nn CPX $nn

功用：累加器 A 中的内容与指定零页单元的数据进行比较，比较后累加器中的内容不变，仅改变标志寄存器的 N、Z、C 位。

说明：CPY $nn 与 CPY $00nn

CMP $nn 与 CMP $00nn

CPX $nn 与 CPX $00nn

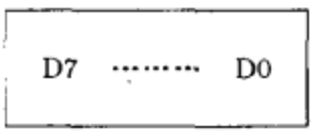
的功能是相同的，请参阅绝对寻址指令的介绍。

第五节 隐含寻址

“隐含寻址”(Implied Addressing)是指操作数已为操作码隐含，不需要在指令中指出操作数的一种寻址方式。

一、隐含寻址方式的指令形式

隐含寻址方式的指令为单字节指令，其指令形式为：



操作码(OP)

其汇编指令形式为：

助记符

例如：指令代码“CLC”(机器码为 18)的功能是清除标志寄存器的进位标志位，即令 C=0。因为清除进位位就是对标志寄存器的进位 C 操作，操作码本身已经包含了操作数的内容，所以可以省掉表示操作数的字节。

再如，“DEX”指令(机器码为 CA)，它执行变址寄存器 X 的内容减 1 操作，即令 X−1→X，操作码中已隐含了对变址寄存器 X 进行操作的信息，同样不需要选用另外的字节来存储操作数。

二、隐含寻址方式说明

1. 隐含寻址方式的指令均为单字节指令，操作数已隐含在操作码中。

2. 隐含寻址方式的操作仅在累加器 A、变址寄存器 X、Y、堆栈寄存器 S、标志寄存器 P 中进行，不能对存储器进行操作。

3. 隐含寻址方式除对寄存器操作外还有三条指令用于中断返回、子程序调用返回和空操作。

4. 因隐含寻址方式指令为一个字节，故占用内存少、执行速度快，编程效率高。

三、隐含寻址方式的指令

隐含寻址方式的指令共有 24 条。

1. 累加器内容进、出栈指令

格式：48 PHA

68 PLA

功用：PHA 指令控制累加器 A 中的内容进栈保存；PLA 指令控制堆栈中保存的数据出栈，并送入累加器 A。

注释：PHA 来源于“PusH Accumulator”，是“累加器 A 中的内容压栈”的意思。PLA 来源于“Pull Accumulator”，是“从堆栈中弹出的内容装入累加器 A”的意思。

说明：PHA、PLA 两条指令互为逆操作，执行后将改变标志寄存器的 N、Z 位。

示例：

```
A9 88   LDA # $88     ;88H→A
48      PHA           ;A 的内容压栈
A9 55   LDA # $55     ;55H→A
68      PLA           ;原压栈的内容出栈
```

执行上述程序后，累加器 A 中的内容不是 55H，而是 88H。同时标志寄存器的 N＝1、Z＝0。

2. 标志寄存器内容进、出栈指令

格式：08 PHP

28 PLP

功用：PHP 指令控制标志寄存器 P 中的内容压栈保存；PLP 指令控制堆栈中保存的数据出栈，并送入标志寄存器 P 中。

注释：PHP 来源于“PusH Processor status”，是“标志寄存器 P 中的内容压栈”的意思。PLP 来源于“Pull Processor status”，是“从堆栈中弹出的内容装入标志寄存器 P”的意思。

说明：PHP、PLP 两条指令互为逆操作，执行后将改变标志寄存器的 N、Z 位。

3. 寄存器之间数据传送指令

格式：AA TAX ；A→X

8A TXA ；X→A

A8 TAY ；A→Y

98 TYA ；Y→A

9A TXS ；X→S

BA TSX ；S→X

功用：完成寄存器之间的数据传送。

注释：TAX 来源于“Tranfer Accumulator to indexregister X”，是“传递累加器 A 中的内容至变址寄存器 X 中”的意思(后面 5 条指令类同)。

示例：

```
A2 56   LDX # $ 56     ;56H→A
8A      TXA            ;X→A
A8      TAY            ;A→Y
```

上述程序执行后，X＝56、A＝56、Y＝56。

4. 变址寄存器增 1 指令

格式：E8 INX ；X＋1→X

C8 INY ；Y＋1→Y

功用：控制变址寄存器中的内容加 1。

说明：以上指令将影响标志寄存器的 N、Z 位。

5. 变址寄存器减 1 指令

格式：CA DEX ；X－1→X

88 DEY ；Y－1→Y

功用：控制变址寄存器中的内容减 1。

说明：以上指令将影响标志寄存器的 N、Z 位。

6. 标志寄存器的进位标志设置、清除指令

格式：38 SEC ；C＝1

18 CLC ；C＝0

功用：对标志寄存器的进位标志位进行操作。

注释：SEC 来源于“SEt Carry”，是“设置进位标志”的意思；CLC 来源于“CLear Carry”，是“清除进位标志”的意思。

说明：以上指令将影响标志寄存器的 C 位。

7. 标志寄存器的十进制运算标志设置、清除指令

格式：F8 SED ；D＝1

D8 CLD ；D＝0

功用：对标志寄存器的十进制运算标志位进行操作。

说明：以上指令将影响标志寄存器的 D 位。

8. 标志寄存器的可屏蔽中断标志设置、清除指令

格式：78 SEI ；I＝1

58 CLI ；I＝0

功用：对标志寄存器的可屏蔽中断标志位进行操作。

说明：以上指令将影响标志寄存器的 I 位。

9. 标志寄存器的溢出标志位清除指令

格式：B8 CLV ；V＝0

功用：清除标志寄存器溢出标志位。

说明：该指令将令标志寄存器的 V＝0。

10. 控制中断程序返回指令(40)

格式：40 RTI

功用：控制中断处理程序返回，恢复中断处理前的现场。

注释：RTI 来源于“ReTurn from Interrupt”，是“从中断返回”的意思。

说明：执行该指令后，标志寄存器将恢复原来的状态。

11. 控制子程序返回指令(60)

格式：60 RTS

功用：控制中断处理程序返回，恢复中断处理前的现场。

说明：该指令不影响标志寄存器。

12. 空操作指令(EA)

格式：EA NOP

功用：系统执行一次空操作。顾名思义，空操作不进行任何操作，仅起到占用固定机时的作用。编程中，一般使用空操作指令控制延时。

第六节　累加器寻址

操作数在累加器 A 中的寻址方式称为累加器寻址(Accumulator Addressing)。

一、累加器寻址方式的指令形式

累加器寻址中，如指令“ROR A”的功能是：累加器 A 中的操作数各位、连同进位 C 一起循环右移一位，使原来的最低位移入进位 C，原来进位 C 移入最高位。由于累加器寻址方式约定操作数放在累加器 A 中，所以可以省掉表示操作数的字节，使累加器寻址为单字节指令。其指令形式为：

操作码(OP)

其汇编指令形式为：

助记符

因此，累加器寻址方式中的循环右移位指令可以表示为“ROR”

二、累加器寻址方式说明

1. 累加器寻址方式指令均为单字节指令。

2. 由于累加器寻址方式指令的操作约定在累加器中，不需要单独指出操作数的地址，即操作数地址隐含在操作码中。因此，从这个角度说，累加器寻址方式也是一种隐含寻址方式。

3. 累加器寻址方式的指令占用空间少、执行速度快、编程效率高。

三、累加器寻址方式的指令

累加器寻址方式的指令仅有 4 条。

1. 左移位指令(0A)

格式：0A ASL

功用：累加器 A 中的内容左移一位，原来的最高位移入进位 C，最低位填入 0。

说明：该指令将影响标志寄存器的 N、Z、C 位。

2. 右移位指令(4A)

格式：4A LSR

功用：累加器 A 中的内容右移一位，原来的最低位移入进位 C，最高位填入 0。

说明：该指令将影响标志寄存器的 N、Z、C 位。

3. 循环左移位指令(2A)

格式：2A ROL

功用：累加器 A 中的内容连同进位 C 一起循环左移一位，原来的最高位移入进位 C，原来的进位移入最低位。

说明：该指令将影响标志寄存器的 N、Z、C 位。

4. 循环右移位指令(6A)

格式：6A ROR

功用：累加器 A 中的内容连同进位 C 一起循环右移一位，原来的最低位移入进位 C，原来的进位移入最高位。

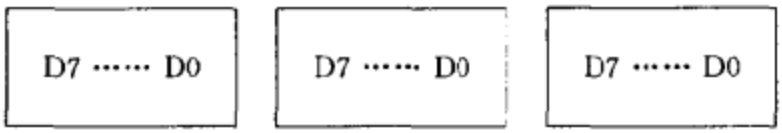
说明：该指令将影响标志寄存器的 N、Z、C 位。

第七节　绝对 X 变址寻址

绝对 X 变址寻址(Absolute X－Indexed Addressing)是使用 X 寄存器进行变址的一种寻址方式，操作数的地址由绝对地址再加上 X 寄存器中的数值构成。

一、绝对 X 变址寻址方式的指令形式

绝对 X 变址寻址方式的指令均为三字节指令，指令形式与绝对寻址方式的指令形式相同。为：



操作码(OP)　　操作数(低位)　　操作数(高位)

其汇编指令形式为：

助记符 $n2n1

例如：指令“BD 00 80”(LDA $8000,X)是一条取数指令，其操作码为“BD”，操作数存放在内存的一个单元中，该单元的地址由指令的第二、三字节指定的地址再加上变址寄存器 X 中的内容得到。

一般将指令中给出的两字节绝对地址称为基地址，将变址寄存器 X 中的内容称为地址偏移量(或偏址指针)，则有：

操作数的实际存放地址＝基地址＋地址偏移量(X)

因此，上述指令的功用就是把 $8000＋X 单元的内容取出来，送入累加器 A 中。若 X＝30H，则把 $8030 中的数据置入 A 中。记为：

($8000+X)→A

二、绝对 X 变址寻址方式说明

1. 绝对 X 变址寻址方式的指令均为三字节。

2. 同一条 X 变址指令，将因 X 的取值不同而改变了操作数的地址(这就是把 X 称为变址寄存器的原因)。

三、绝对 X 变址寻址方式的指令

绝对 X 变址寻址方式共有 15 条指令，其中取数指令 2 条、存数指令 1 条、算术运算指令 2 条、存储器增、减指令 2 条、逻辑运算指令 3 条、位移指令 4 条、比较指令 1 条。

1. 取数指令

格式：BC n1 n2 LDY $n2n1,X ;($n2n1+X)→Y

BD n1 n2 LDA $n2n1,X ;($n2n1+X)→A

功用：取($n2n1+X)单元的数据送入 A 或 Y 寄存器。

说明：以上指令将改变标志寄存器的 N、Z 位。

2. 存数指令

格式：9D n1 n2 STA $n2n1,X ;A→($n2n1+X)

功用：把累加器 A 中的数据存入($n2n1+X)单元。

3. 算术加、减运算指令

格式：7D n1 n2 ADC $n2n1,X ;带进位加

FD n1 n2 SBC $n2n1,X ;带进位减

功用：累加器带进位加、减($n2n1+X)单元的内容，结果仍存在累加器中。

说明：以上指令将改变标志寄存器的 N、V、Z、C 位。

4. 存储器增、减 1 指令

格式：FE n1 n2 INC $n2n1,X ;($n2n1+X)+1

DE n1 n2 DEC $n2n1,X ;($n2n1+X)−1

功用：存储器中的内容加、减 1。

说明：以上指令将改变标志寄存器的 N、Z 位。

5. 逻辑运算指令

格式：1D n1 n2 ORA $n2n1,X ;A OR ($n2n1+X)

3D n1 n2 AND $n2n1,X ;A AND ($n2n1+X)

5D n1 n2 EOR $n2n1,X ;A EOR ($n2n1+X)

功用:累加器中的数据与存储器中的数据进行逻辑运算,结果仍存在累加器中。

说明:以上指令将改变标志寄存器的 N、Z 位。

6. 位移指令

格式:1E n1 n2 ASL $n2n1,X ;($n2n1+X)左移位

5E n1 n2 LSR $n2n1,X ;($n2n1+X)右移位

3E n1 n2 ROL $n2n1,X ;($n2n1+X)左转位

7E n1 n2 ROR $n2n1,X ;($n2n1+X)右转位

功用:进行存储器移位操作。

说明:以上指令将改变标志寄存器的 N、Z、C 位。

7. 比较指令

格式:DD n1 n2 CMP $n2n1,X ;A-($n2n1+X)

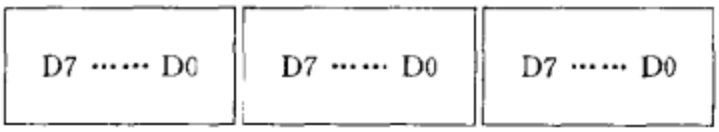
功用:比较累加器 A 中与($n2n1+X)单元中的数据,改变标志寄存器的 N、Z、C 位。

第八节 绝对 Y 变址寻址

绝对 Y 变址寻址(Absolute Y-Indexed Addressing)是使用 Y 寄存器进行变址的一种寻址方式,操作数的地址由绝对地址再加上 Y 寄存器中的数值构成。

一、绝对 Y 变址寻址方式的指令形式

绝对 Y 变址寻址方式的指令均为三字节指令，指令形式与绝对寻址方式的指令形式相同。为：



操作码(OP)　　操作数(低位)　　操作数(高位)

其汇编指令形式为：

助记符 $n2n1

例如：指令“BE 00 80”(LDX $8000,Y)是一条取数送入变址寄存器 X 的指令，其操作是以指令中给出的两字节绝对地址为基地址，以变址寄存器 Y 中的内容为地址偏移量，取出“基地址＋地址偏移量(Y)”单元的数据送入 X。上述指令中若 Y＝50，则把 $8050 中的数据置入 X 中。记为：

($8000＋Y)→X

二、绝对 Y 变址寻址方式说明

1. 绝对 Y 变址寻址方式的指令均为三字节。

2. 绝对 Y 变址与绝对 X 变址寻址方式十分相似，其区别仅是前者使用 Y 变址、后者使用 X 变址。

3. 绝对 Y 变址各指令操作后对标志寄存器各位的影响，与绝对 X 变址寻址方式相同。

三、绝对 Y 变址寻址方式的指令

绝对 Y 变址寻址方式共有 9 条指令，其中取数指令 2 条、存数指令 1 条、算术运算指令 2 条、逻辑运算指令 3 条、比较指令 1 条。

1. 取数指令

格式：B9 n1 n2 LDA $n2n1,Y ;($n2n1+Y)→A

BE n1 n2 LDX $n2n1,Y ;($n2n1+Y)→X

功用：取($n2n1+Y)单元的数据送入A或X寄存器。

2. 存数指令

格式：99 n1 n2 STA $n2n1,Y ;A→($n2n1+Y)

功用：把累加器A中的数据存入($n2n1+Y)单元。

3. 算术加、减运算指令

格式：79 n1 n2 ADC $n2n1,Y ;带进位加

F9 n1 n2 SBC $n2n1,Y ;带进位减

功用：累加器带进位加、减($n2n1+Y)单元的内容，结果仍存在累加器中。

4. 逻辑运算指令

格式：19 n1 n2 ORA $n2n1,Y ;A OR ($n2n1+Y)

39 n1 n2 AND $n2n1,Y ;A AND ($n2n1+Y)

59 n1 n2 EOR $n2n1,Y ;A EOR ($n2n1+Y)

功用：累加器中的数据与存储器中的数据进行逻辑运算，结果仍存在累加器中。

5. 比较指令

格式：D9 n1 n2 CMP $n2n1,Y ;A-($n2n1+Y)

功用：比较累加器A中与($n2n1+Y)单元中的数据，改变标志寄存器的N、Z、C位。

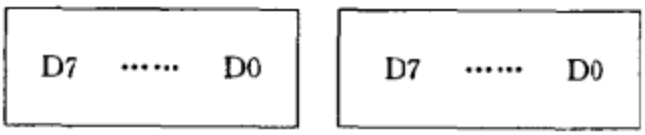
第九节 零页变址寻址

使用变址寄存器对零页进行变址寻址的方式称为零页变址寻址。

零页变址寻址有两种方式：以变址寄存器 X 变址的称为零页 X 变址寻址(Zero－page X－indexed addressing)；以变址寄存器 Y 变址的称为零页 Y 变址寻址(Zero－page Y－indexed addressing)。

一、零页变址寻址方式的指令形式

零页变址寻址方式为两字节指令，其指令形式如下：



操作码(OP)　　操作数(低位)

其汇编指令形式为：

助记符 $nn

例如：取数指令“LDA $FF,X”(机器码为 B5 FF)的功用是把地址为“$00FF＋X”单元中的数据取出送入累加器 A 中。

二、零页变址寻址方式说明

1. 零页变址寻址方式的指令均为两个字节。

2. 零页变址寻址与绝对变址寻址方式的含义基本相同，仅在寻址范围上有差异。前者仅限于零页寻址，后者可达 64K 字节。

3. 零页变址寻址只能产生零页地址，超过十六进制两位数的部分应丢掉。如：当 X＝D8 时，指令“LDA $80，X”得到的有效地址是 $58，而不是 $158。

4. 零页变址寻址方式指令操作后，对标志寄存器各位的影响情况与绝对变址寻址方式相同。

三、零页 X 变址寻址指令

零页 X 变址寻址方式共有 16 条指令。其中：取数指令 2 条、存数指令 2 条、算术运算指令 2 条、加 1 减 1 指令 2 条、逻辑运算指令 3 条、移位指令 4 条、比较指令 1 条。

1. 取数指令

格式：B4 nn LDY $nn,X ;($nn+X)→Y

B5 nn LDA $nn,X ;($nn+X)→A

功用：取零页($nn+X)单元的数据送入 A 或 Y 寄存器。

2. 存数指令

格式：84 nn STY $nn,X ;Y→($nn+X)

85 nn STA $nn,X ;A→($nn+X)

功用：把寄存器中的数据存入($nn+X)单元。

3. 算术加、减运算指令

格式：75 nn ADC $nn,X ;带进位加

F5 nn SBC $nn,X ;带进位减

功用：累加器带进位加、减($nn+X)单元的内容，结果仍存在累加器中。

4. 存储器增、减 1 指令

格式：F6 nn INC $nn,X ;($nn+X)+1

D6 nn DEC $nn,X ;($nn+X)−1

功用：零页存储器中的内容加、减 1。

5. 逻辑运算指令

格式：15 nn ORA $nn,X ;A OR ($nn+X)

35 nn AND $nn,X ;A AND ($nn+X)

55 nn EOR $nn,X ;A EOR ($nn+X)

功用：累加器中的数据与零页存储器中的数据进行逻辑运算，结果仍存在累加器中。

6. 位移指令

格式：16 nn ASL $nn,X ;($nn+X)左移位

56 nn LSR $nn,X ;($nn+X)右移位

36 nn ROL $nn,X ;($nn+X)左转位

76 nn ROR $nn,X ;($nn+X)右转位

功用：进行零页存储器的移位操作。

7. 比较指令

格式：D5 nn CMP $nn,X ;A－($nn+X)

功用：比较累加器 A 中与($nn+X)单元中的数据，改变标志寄存器的 N、Z、C 位。

四、零页 Y 变址寻址方式的指令

零页 Y 变址寻址方式仅有 2 条指令。

1. 取数指令

格式：B6 nn LDX $nn,Y ;($nn+Y)→X

功用：取零页($nn+Y)单元的数据送入 X 寄存器。

说明：该指令将改变标志寄存器的 N、Z 位。

2. 存数指令

格式：96 nn STX $nn,Y ;X→($nn+Y)

功用：把 X 寄存器中的数据存入($nn+Y)单元。

第十节 间接寻址

在指令的操作数部分，若给出的是操作数的间接地址，那么这种寻址方式就称为间接寻址(Indirect Addressing)。

在间接寻址指令中，操作数部分给出的是操作数所在地址的地址的低位。也就是说，在操作码后面两个字节中的地址，并不是有效地址，而是有效地址的低位存储单元，有效地址的高位则在低位单元的下一个单元。因此，间接寻址的有效地址必须经过两次寻址才能找到。

例如：若 $8000 单元存放的数据为 98H、$8001 单元存放的数据是 FAH，对于间接寻址的无条件转移指令“JMP ($8000)”来说，首先从间接地址 $8000 中取出有效地址的低位 98H、再从间接地址加 1 单元 $8001 中取出有效地址的高位 FAH，则 $FA98 才是操作数的有效地址。因此，执行指令

JMP ($8000)

的操作就是转移到地址为 $FA98 处继续执行。

注意：

JMP $8000

JMP ($8000)

两条指令之间的差别。虽然它们都是无条件转移指令，但它们的操作码不同。前者的操作码为“4C”，是绝对寻址指令，有点像 BASIC 语言中的

GOTO 8000

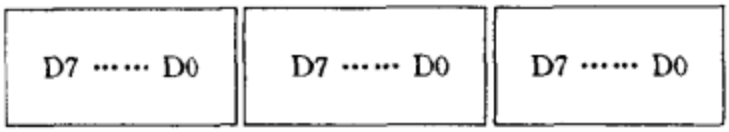
语句；后者的操作码为“6C”，是间接寻址指令，它的操作若用 BASIC 语言表示，应为

GOTO 8000

8000 GOTO FA98

即“8000”相当于程序行号，8000 行的目标行号“FA98”才是转移的目标地址。

间接寻址为 3 字节指令，指令形式为



操作码(OP)　操作数地址的地址（低位）　操作数地址的地址（高位）

间接寻址方式的指令只有1条。

格式：4C n1 n2 JMP（$n2n1）

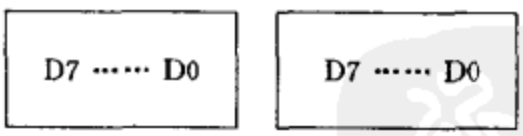
功用：从$n2n1单元取出地址地位、从$n2n1＋1单元取出地址高位，控制系统无条件地转移到指定地址继续执行。

第十一节　相对寻址

以指令当前的地址为参照点，向前、向后跨越一定偏移量的寻址方式称为相对寻址(Relative Addressing)。

一、相对寻址的指令形式

相对寻址为两字节指令，其形式为：



操作码(OP) 操作数地址的相对偏移量

指令中第一字节为操作码，第二字节为相对转移地址。相对转移地址又称“相对位移量”或“偏移量D”，它实际上就是条件转移指令的跳转步长。D值可正、可负，正、负由最高位指明，若最高位等于1则为负值，将向前跳转；若最高位等于0则为正值，将向后跳转。操作数的实际地址是以本指令的下一条指令地址为基地址，再加偏移量D而得来的。

二、相对寻址负跳转操作分析

为了加深对相对寻址方式的认识，我们先看一个程序实例。例 2－1 设计一个把存储器二页所有单元清零的子程序。程序的起始地址在 $8000。

程序见 No. 2－1。

No. 2－1

```
8000 A9 00     LDA  # $00      ;A=0
8002 A8        TAY            ;Y=0
8003 99 00 02  STA  $0200,Y   ;0→$200+Y
8006 C8        INY            ;Y=Y+1
8007 D0 FA     BNE  $8003     ;若 Y≠0 转 $8003
8009 60        RTS            ;若 Y=0 返回
```

程序 No. 2－1 各行程序的解释如下：

1. 第一行为立即寻址，累加器 A 取 0 值；

2. 第二行为隐含寻址，累加器 A 的数据送入变址寄存器 Y，Y＝0；

3. 第三行为绝对 Y 变址寻址方式，A＝0→（$200＋Y）单元，当前 Y＝0，故有 0→$200 单元，即 $200＝0；

4. 第四行为隐含寻址，Y＝Y＋1，即 Y＝1；

5. 第 5 行就是相对寻址指令。“D0”为操作码，“FA”为偏移量，该指令的功用是判断标志寄存器的 Z 标志，若 Z＝0 则条件满足，执行本指令的操作；若 Z＝1 则条件不满足，跳过本指令执行下一条指令。

在本指令中，偏移量 D＝FA，因其最高位等于 1，所以为负值，程序应向前跳转，向前跳转地址的计算方法如下：

(1)取出下一条指令的地址 $8009 为基地址；

(2)取出偏移量 D,因 D=FA、是负值(负值用补码 表示),应还原为原码,即令 FA−1 后再按位取 反,得到偏移量的原码形式为 06;

(3)令基地址减偏移量,有

8009−6=8003

$8003 即为程序跳转的实际地址。

所以,程序将由第 5 行控制向前跳转到 $8003,即第 3 行继续向二页单元置数。由于现在 Y=1,故令 $201=0。

6. 如此循环不已,对二页的置 0 操作一直进行下去。直到对二页的最后一个单元 $2FF 置入 0 后,在执行第 4 行令 Y 加 1 时,因 Y=FF+1=0(Y 为 8 位寄存器,自动舍去进位),标志寄存器的 Z 标志位为 1,不满足第 5 行的条件,故执行第 6 行程序,结束操作,返回调用程序。

三、相对寻址正跳转操作分析

以上为负跳转的情况,下面再以例 2−1 编写一个正跳转的程序,程序见 No. 2−2。

No. 2−2

```
8000 A9 00     LDA  # $00      ;A=0
8002 A8        TAY             ;Y=0
8003 99 00 02  STA  $0200,Y    ;0→$200+Y
8006 C8        INY             ;Y=Y+1
8007 F0 03     BEQ  $800C      ;若 Y=0 转 $800C
8009 4C 03 80  JMP  $8003      ;若 Y≠0 转 8003
800C 60        RTS
```

程序 No. 2−2 的第 5 行程序是相对寻址指令,它的执行条件是当标志寄存器的 Z 标志位等于 1,即 Y 的取值为 0 时,将跳转到 $800C 行。若 Y 的取值不等于 0 则跳过本行,执行

第六行程序，无条件地转移到$8003行继续进行对二页单元的置0操作。

当Y等于0时，第5行条件满足，取出下一行程序的首址$8009为基地址；取出偏移量D(D=3)，因D的最高位为0，为正数，应向后跳转，令基地址加偏移量，有

8009+3=800C

因此，程序将跳转到$800C，执行子程序返回操作。

四、相对寻址中程序计数器的操作

明白了以上相对寻址方式的跳转计算后，我们再来分析跳转中程序计数器PC的作用(以程序No.2—2为例)。

1. 上述程序执行时，PC中首先置入的是程序首地址$8000。于是，电脑依据PC的值，取出$8000地址的指令“A9 00”，同时令该指令的长度(字节数2)与PC值相加(8000+2=8002)，再送入PC。此时，PC的值为8002，正好是第二行程序的首地址。

2. 第一条指令执行完毕，依PC的值取指令“A8”，并令PC+1=8003。

3. 第二条指令执行完毕，依PC的值取指令“99 00 02”，并令PC+3=8006。

4. 第三条指令执行完毕，依PC的值取指令“C8”，并令PC+1=8007。

5. 第四条指令执行完毕，依PC的值取指令“F0 03”，并令PC+2=8009。

“F0 03”指令要求检查其上一条指令执行后，Z标志寄存器的状态(Y=0时Z=1；Y≠0时Z=0)，当Z=1时条件满足，程序将产生转移。于是，将PC中的值(8009)加上偏移

量(03)，再置入 PC 中。现在 PC＝800C，因此，将从 800C 取出指令“60”，执行子程序返回操作。

五、相对寻址方式说明

1. 相对寻址指令均为两个字节。

2. 相对寻址只适用于条件转移指令。

3. 相对寻址指令的操作数部分是转移指令的跳转步长。

4. 相对寻址指令中，操作数的实际地址是取出本指令后的 PC 值与偏移量之和。

5. 偏移量为负数时，要用补码表示。

6. 为简化实际编程操作，书写相对寻址指令的助记符时，用转移后的目标地址代替偏移量。

六、相对寻址方式的指令

相对寻址方式共有 8 条指令。其中 Z 标志判断转移指令 2 条、C 标志判断转移指令 2 条、N 标志判断转移指令 2 条、V 标志判断转移指令 2 条。

1. Z 标志判断转移指令

格式：D0 nn BNE $nn ；非零转移

F0 nn BEQ $nn ；零转移

功用：根据标志寄存器的 Z 标志决定程序是否转移。当操作数非零时，Z＝0、满足非零转移条件；当操作数为零时，Z＝1、满足零转移条件。

2. C 标志判断转移指令

格式：90 nn BCC $nn ；无进位转移

B0 nn BCS $nn ；有进位转移

功用：根据标志寄存器的 C 标志决定程序是否转移。当

C=0 时满足无进位转移条件；当 C=1 时满足有进位转移条件。

3. N 标志判断转移指令

格式：10 nn BPL $nn ；正转移

30 nn BMI $nn ；负转移

功用：根据标志寄存器的 N 标志决定程序是否转移。当操作数为正数(<80H)时，N=0、满足正转移条件；当操作数为负数时(>7FH)，N=1、满足负转移条件。

4. V 标志判断转移指令

格式：50 nn BVC $nn ；标志位 V=0 转移

70 nn BVS $nn ；标志位 V=1 转移

功用：根据标志寄存器的 V 标志决定程序是否转移。当没有溢出时，V=0；当发生溢出时，V=1。

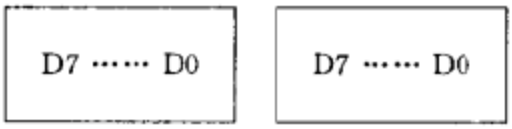
第十二节　先变址间接寻址

先变址间接寻址方式，是先以变址寄存器 X 变址、然后进行间接寻址的，实际上它是零页 X 变址和间接寻址两种方式的结合，因此一般称为先变址(X)间接寻址((IND,X))，简称“X 变间寻址”。

首先，它对零页基地址(IND)进行 X 变址，求得零页地址 IND+X；然后，再以 IND+X 作为间接地址，经两次间接寻址找到操作数的有效地址。其中第一次对 IND+X 间址得到的是操作数有效地址的低 8 位；第二次对 IND+X+1 间址得到操作数有效地址的高 8 位。

一、先变址(X)间接寻址的指令形式

先变址(X)间接寻址方式的指令为两字节指令，其形式为：



操作码(OP)　　操作数(零页基地址)

汇编指令形式为：

助记符（$nn,X)

在操作数($nn,X)中，“$nn，X”在圆括号之内是先进行X变址的标志；圆括号则是间接寻址的标志。

例如，若已知X＝2、$08＝78、$09＝F8、$F878＝54，取数指令

LDA（$06,X)

由指令可知，IND＝06，根据上述定义，先计算

IND+X＝06+02＝08

这是该指令中操作数的第一次间址，从$08单元得到有效地址的低8位78H；第二次间址，计算

IND+X+1＝06+02+1＝09

从$09单元得到有效地址的高8位F8H；所以，操作数的实际地址就是$F878。本指令的功用就是把$F878单元的数据54H送入累加器A中。

二、先变址(X)间接寻址指令说明

1. X变间寻址方式的指令均为两字节指令。
2. 变址寄存器只能使用X寄存器。
3. 先做X零页变址，再两次间接寻址，才能得到操作数

的有效地址。

三、先变址(X)间接寻址方式的指令

X 变间寻址共有 8 条指令。其中取数指令 1 条、置数指令 1 条、算术运算指令 2 条、逻辑运算指令 3 条、比较指令 1 条。

1. X 变间寻址方式的取数指令

格式:A1 nn LDA ($nn,X)

功用:取以 X 先变址、后间接寻址的存储器中的数据送入累加器 A。

2. X 变间寻址方式的置数指令

格式:81 nn STA ($nn,X)

功用:把累加器 A 中的数据,送入以 X 先变址、后间接寻址的存储器中。

3. X 变间寻址方式的算术运算指令

格式:61 nn ADC ($nn,X) ;带进位加

E1 nn SBC ($nn,X) ;带进位减

功用:取累加器 A 中的数据,与以 X 先变址、后间接寻址的存储器中的数据进行带进位加、减运算,结果仍存在累加器 A 中。

4. X 变间寻址方式的逻辑运算指令

格式:01 nn ORA ($nn,X) ;A OR ($nn,X)

21 nn AND ($nn,X) ;A AND ($nn,X)

41 nn EOR ($nn,X) ;A EOR ($nn,X)

功用:取累加器 A 中的数据,与以 X 先变址、后间接寻址的存储器中的数据进行逻辑运算,结果存在累加器 A 中。

5. X 变间寻址方式的比较指令

格式:C1 nn CMP (＄nn,X);A—(＄nn,X)

功用:先取累加器 A 中的数据,去减以 X 先变址、后间接寻址的存储器中的数据,设置标志寄存器中的 N、Z、C 标志。

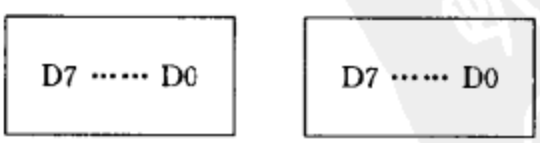
第十三节　后变址间接寻址

后变址间接寻址方式,先在零页间接寻址、然后用 Y 变址寻址,实际上它是零页间接寻址和绝对 Y 变址相结合的一种寻址方式。一般称为后变址(Y)间接寻址方式((IND),Y),简称"Y 间变寻址"。

Y 间变寻址方式的操作是:首先,在零页进行间接寻址,取得一个 16 位的基地址(IND);然后,再进行绝对 Y 变址寻址。即以基地址加 Y 变址寄存器中的内容,得到操作数的实际存放地址。所以这种寻址方式称为先间接寻址后绝对 Y 变址。

一、后变址(Y)间接寻址的指令形式

Y 间变寻址方式的指令为两字节指令,其形式为:



操作码(OP)　操作数(零页基地址)

汇编指令形式为:

助记符 (＄nn),Y

操作数中,"(＄nn)"表示先间接寻址;圆括号外边的"Y"表示后进行 Y 绝对变址寻址。

例如,若已知 Y＝2、＄06＝78、＄07＝F8、＄F87A＝

DD，对取数指令

LDA（$06，Y）

得到基地址为$F878（$06＝78、$07＝F8），根据上述定义，操作数的有效地址应为

IND＋Y＝F878＋02＝F87A

因此，操作数的实际地址就是$F87A。本指令的功用就是把$F87A单元的数据DDH送入累加器A中。

二、后变址（Y）间接寻址指令说明

1. Y间变寻址方式的指令均为两字节指令。

2. 变址寄存器只能使用Y寄存器。

3. 寻址方法是，现在零页中作间接寻址，然后再进行绝对Y变址寻址。

三、后变址（Y）间接寻址方式的指令

Y间变寻址方式共有8条指令。其中取数指令1条、置数指令1条、算术运算指令2条、逻辑运算指令3条、比较指令1条。

1. Y间变寻址方式的取数指令

格式：B1 nn LDA（$nn），Y

功用：取先在零页间接寻址、后进行Y绝对变址的存储器中的数据送入累加器A。

2. Y间变寻址方式的置数指令

格式：91 nn STA（$nn），Y

功用：把累加器A中的数据，送入先在零页间接寻址、后进行Y绝对变址的存储器中

3. Y间变寻址方式的算术运算指令

格式：71 nn ADC（$nn），Y ；带进位加

F1 nn SBC（$nn），Y ；带进位减

功用：取累加器 A 中的数据，与先在零页间接寻址、后进行 Y 绝对变址的存储器中的数据进行带进位加、减运算，结果仍存在累加器 A 中。

4. Y 间变寻址方式的逻辑运算指令

格式：11 nn ORA（$nn），Y ；A OR（$nn），Y

31 nn AND（$nn），Y ；A AND（$nn），Y

51 nn EOR（$nn），Y ；A EOR（$nn），Y

功用：取累加器 A 中的数据，与先在零页间接寻址、后进行 Y 绝对变址的存储器中的数据进行逻辑运算，结果存在累加器 A 中。

5. Y 间变寻址方式的比较指令

格式：D1 nn CMP（$nn），Y ；A－（$nn），Y

功用：先取累加器 A 中的数据，去减先在零页间接寻址、后进行 Y 绝对变址的存储器中的数据，设置标志寄存器中的 N、Z、C 标志。

对汇编语言刚入门的朋友，往往容易混淆“X 变间、Y 间变”两种寻址方式的概念。其实它们很好区别，开始只要记住：“X 先变后间、Y 先间后变”十个字就可以了，以后随着编程实践的增多，就会逐渐加深对它们的认识，进入运用自如的境界。

第三章　6527的指令系统

指令系统是程序设计的基础，任何一个程序设计都是在一定的指令系统下完成的。

程序总是以完成预定任务为目的的，就其组成来说，主要包括算术、逻辑运算和各种控制操作，而这些操作的目标就是各种数据和代码。因此，任何一个指令系统都应包括：数据传送、程序控制和算术、逻辑运算四大类指令。

我们知道，6527微处理器共有55条基本指令，通过与13种寻址方式组合，可延伸出151条操作指令。所以，我们可以认为6527微处理器共有151条指令。

第二章我们以寻址方式为线，了解了151条指令的使用格式、功用和操作特点。为加深认识，本章将以功能为纲，从使用角度出发，重复介绍这151条指令。希望通过本章的学习，能使大家更进一步熟悉6527的指令系统、了解6527编程中所使用的寄存器、掌握各种寻址方式的特点、熟悉每条指令的功能以及对标志寄存器各位所产生的影响等。

第一节　数据传送类指令

数据的传送是计算机进行的最基本、最经常、最频繁的操作，它在程序中占有很大的比重。数据的传送是否灵活、传送速度是否快捷、寻址方式的选用是否合理等方面，都对程序的结构、运行速度、工作效率产生极大的影响。因此，掌握

数据的传送操作，是学习6527机器语言程序设计的基本功。

6527指令系统的数据传送类指令主要包括：

数据由存储器取到寄存器

数据由寄存器传送到存储器(内存单元)

各寄存器之间的数据传送

三大部分，共20条指令。分为取数、送数、传送、进出栈、移位5种类型。

一、取数指令

取数指令的功用是令A、X、Y三个寄存器获得数据。它有3条基本指令“LDA、LDX、LDY”。指令执行后只影响标志寄存器的N、Z位，如果传送数据的最高位是1则置P寄存器的N位为1，最高位是0则置N为0，因此N又称为“符号位”；如果传送数据的各位是0则置Z位为1，否则置Z位为0，因此Z又称“零标志位”。

1. LDA——取数送累加器A

根据寻址方式的不同，LDA指令共有8种形式(见表3—1)。LDA指令的功用是将一个立即数或某个存储单元的内容送入累加器A中。

例如，“LDA ＃＄98”是立即寻址方式的指令，其机器码为“A9 98”。执行该指令后，除了立即数98送入累加器A外，还对P寄存器的N、Z位产生影响。因传送的数据最高位为1(98H＝10011000B)，认为数据是负数，故N＝1；因传送的数据不为0，故Z＝0。

至于程序计数器PC中的内容，将由当前指令的地址n2n1(LDA ＃＄98的首地址处)，指向下一条指令的首地址n2n1＋2。

表 3－1　LDA 指令寻址方式一览表

指令格式	功　用	机器码	寻址方式
LDA #$n	#$n → A	A9 n	立即寻址
LDA $n	($n) → A	A5 n	零页寻址
LDA $n,X	($n+X) → A	B5 n	零页 X 变址
LDA $n2n1	($n2n1) → A	AD n1 n2	绝对寻址
LDA $n2n1,X	($n2n1+X)→A	BD n1 n2	绝对 X 变址
LDA $n2n1,Y	($n2n1+Y)→A	B9 n1 n2	绝对 Y 变址
LDA ($n,X)	(地址) → A	A1 n	X 变间寻址
LDA ($n),Y	(地址) → A	B1 n	Y 间变寻址

2. LDX——取数送变址寄存器 X

LDX 指令的功用是将一个立即数或某个存储器中的数据送入变址寄存器 X。它有 5 种寻址方式(见表 3－2)。

表 3－2　LDX 指令寻址方式一览表

指令格式	功　用	机器码	寻址方式
LDX #$n	$n → X	A2 n	立即寻址
LDX $n	($n) → X	A6 n	零页寻址
LDX $n,Y	($n+Y) → X	B6 n	零页 Y 变址
LDX $n2n1	($n2n1) → X	AE n1 n2	绝对寻址
LDX $n1n2,Y	(n2n1+Y)→ X	BE n1 n2	绝对 Y 变址

例如，指令“LDX $8000,Y”，采用 Y 绝对变址寻址方式。设 Y＝30、$8030＝48，则指令执行后，将取绝对地址 $8000 加 Y 寄存器的偏移量，得到变址 $8030，取出 $8030 单元的数据 48 置入 X 寄存器。因 48(01001000)的最高位等于 0，所以 P 寄存器的 N＝0、Z＝0。由于绝对 Y 变址是 3 字节指令，所以程序计数器 PC 指向本条指令首地址＋3。

3. LDY——取数送变址寄存器 Y

LDY 指令的功用与 LDX 类似，也是将一个立即数或某个存储器中的数据送入变址寄存器 Y。它也有 5 种寻址方式（见表 3—3）。

表 3—3 LDY 指令寻址方式一览表

指令格式	功 用	机器码	寻址方式
LDY # $n	$n → Y	A0 n	立即寻址
LDY $n	($n) → Y	A4 n	零页寻址
LDY $n,X	($n+X) → Y	B4 n	零页 X 变址
LDY $n2n1	($n2n1) → Y	AC n1 n2	绝对寻址
LDY $n1n2,X	(n2n1+X)→Y	BC n1 n2	绝对 X 变址

二、送数指令

送数指令的功用是指 CPU 将某个寄存器中的数据送入存储器的某个单元中。它有 3 条基本指令：STA、STX、STY。各指令执行后对标志位不产生影响。

1. STA——累加器 A 的数据送入存储器

STA 指令执行后，累加器 A 中的内容不变，指定内存中的内容改变。例如，

```
LDA # $33
STA $86
```

执行后，零页 $86 单元的内容为 33。

STA 有 7 种寻址方式，见表 3—4。

表 3－4　STA 指令寻址方式一览表

指令格式	功　用	机器码	寻 址 方 式
STA $n	A→($n)	85 n	零页寻址
STA $n,X	A→($n+X)	95 n	零页 X 变址
STA $n2n1	A→($n2n1)	8D n1 n2	绝对寻址
STA $n2n1,X	A→(n2n1+X)	9D n1 n2	绝对 X 变址
STA $n2n1,Y	A→(n2n1+Y)	99 n1 n2	绝对 Y 变址
STA ($n,X)	A→(地址)	81 n	X 变间寻址
STA ($n),Y	A→(地址)	91 n	Y 间变寻址

2. STX——变址寄存器 X 中的数据送入存储器

STX 指令执行后，X 中的内容不变，指定内存中的内容改变。STX 的指令中不允许使用 X 变址，即 STX $nn,Y 是合法的，但 STX $nn,X 则是非法的。STX 有 3 种寻址方式（见表 3－5）。

表 3－5　STX 指令寻址方式一览表

指令格式	功　用	机器码	寻 址 方 式
STX $n	X→($n)	86 n	零页寻址
STX $n,Y	X→($n+Y)	96 n	零页 Y 变址
STX $n2n1	X→($n2n1)	8E n1 n2	绝对寻址

3. STY——变址寄存器 Y 中的数据送入存储器

STY 指令执行后，Y 中的内容不变，指定内存中的内容改变。STY 的指令中不允许使用 Y 变址，即 STY $nn,X 是合法的，但 STY $nn,Y 则是非法的。STY 有 3 种寻址方式（见表 3－6）。

表 3—6　STY 指令寻址方式一览表

指令格式	功　用	机器码	寻址方式
STY $n	Y → ($n)	84 n	零页寻址
STY $n,X	Y → ($n+X)	94 n	零页 X 变址
STY $n2n1	Y → ($n2n1)	8C n1 n2	绝对寻址

三、寄存器间传送指令

传送指令的功用是在 CPU 内部的寄存器之间传送数据。这类指令均采用隐含寻址方式，指令的操作数隐含在操作码之中，因而均为单字节指令。传送指令共有 6 条(见表 3—7)，除 TXS 指令外，其它 5 条指令执行时将影响标志寄存器的 Z、N 位。

表 3—7　传送指令寻址方式一览表

指令格式	功　用	机器码	寻址方式
TXA	X → A	8A	〃
TAX	A → X	AA	隐含寻址
TYA	Y → A	98	〃
TAY	A → Y	A8	〃
TXS	X → S	9A	〃
TSX	S → X	BA	〃

四、堆栈指令

堆栈指令的功用是在寄存器与堆栈之间传送数据。由寄存器向堆栈区存数叫进栈或压入；由堆栈向寄存器送数叫出栈或弹出。

1. 什么叫堆栈

计算机中的“栈”是用来存放数据的。它是在随机存储器

RAM 中开辟的一个特殊存储区(即从一个固定地址开始的一片连续的内存单元)。在这个存储区中,数据的存(入)、取(出)必须遵循"先进后出"或"后进先出"的原则。这就像货栈一样,最先放入的货物,总是放在最底下,后来的货物依次向上堆;取货时,必须先把放在上面的货物取走(后进先出)后,最后才能取最底下的货物(先进后出)。计算机中把这种特殊结构的存储区叫做"堆栈"。

2. 堆栈底和堆栈顶

6527 统一约定堆栈区设置在第 1 页,即存储器地址为 $100～$1FF 的 256 个单元。堆栈中,数据的存放是从地址码的高单元依次向低单元进行的。$1FF 单元称为"堆栈底",最后存入数据的那个单元称为"堆栈顶"。

3. 堆栈指针

6527 把堆栈顶上面的一个单元地址置入 S 寄存器作为堆栈指针,即由 S 指出新压栈数据的存放地址(因系统约定 1 页为堆栈区,故堆栈的高位地址可以省略,只要指出低位地址就可以了,所以使用一个寄存器 S 就可以指示堆栈)。由于堆栈指针 S 表明了堆栈存、取数据操作的当前位置,所以 S 在数据进栈时有自动减 1 的功能(栈顶上移。进栈时,数据根据 S 先进栈,然后 S 再减 1);在数据出栈时有自动加 1 的功能(栈顶下移。出栈时,S 先加 1,再根据 S 值数据出栈)。由此可见,栈顶指针 S 是随着数据的进、出栈而上、下浮动的。当栈顶地址为 $1FF 时(即 S=FF)则称为空栈。栈顶地址可通过 6527 指令进行预先设置。

图 3—1 中,(A)表示了堆栈中原有 3 个数据 X1、X2、X3 的状态;(B)表示了压入数据 Y1 的状态;(C)表示压入数据 Y2 的状态;(D)表示数据 Y2 从堆栈弹出的状态。

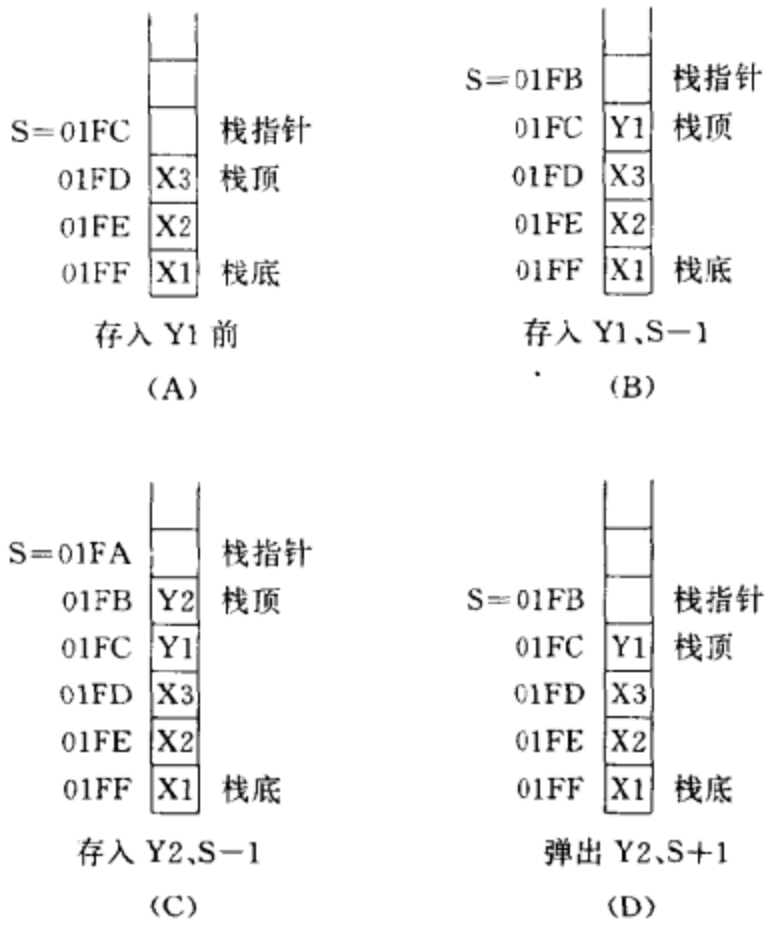


图 3−1　进、出栈中堆栈指针变化情况示意图

4. 进栈、出栈指令

“进栈”是指将寄存器的内容送到堆栈指针所指的单元；“出栈”是将堆栈指针所指向的存储单元的内容送入寄存器。

进、出栈指令只有 4 条，分别为 PHA、PLA、PHP、PLP。它们都采用隐含寻址方式，均为单字节指令。

(1) PHA——累加器 A 进栈指令

把累加器 A 中的内容送入栈顶指针 S 所指向的栈顶空单元中，然后 S 自动减 1。本指令不影响标志寄存器。

(2) PHP——标志寄存器 P 进栈指令

把标志寄存器 P 中的内容送入栈顶指针 S 所指向的栈顶空单元中，然后 S 自动减 1。本指令也不影响标志寄存器。

(3) PLA——累加器 A 出栈指令

它先令栈指针 S 加 1，得到栈顶地址，然后从栈顶中取出数据送入累加器 A。执行本指令将影响标志寄存器的 N、Z 位。

(4) PLP——标志寄存器 P 出栈指令

它先令栈指针 S 加 1，得到栈顶地址，然后从栈顶中取出数据送入标志寄存器 P。由于标志寄存器置入了新的内容，所以本指令将使 P 的内容换为新出栈的内容。

5. 进、出栈指令说明

(1) 进栈指令 PHA、PHP 主要用于保护现场，即保护寄存器 A、P 的状态。

(2) 出栈指令 PLA、PLP 主要用于恢复现场，即恢复寄存器 A、P 原来的状态。

(3)PHA 指令要与 PLA 指令配对使用。

(4) PHP 指令要与 PLP 指令配对使用。

(5) 标志寄存器 X、Y 没有堆栈操作指令，它们的进、出栈操作只能通过累加器 A 间接进行。

五、移位指令

移位指令的功用是将累加器或存储单元中的数据各位依次左、右移动或循环左、右移动。

移位指令有 4 类，分别为：左移位(ASL)、右移位(LSR)、循环左移(ROL)、循环右移(ROR)。每类移位指令都有 5 种寻址方式，表 3－8 仅列出左移位指令的 5 种寻址方式，其它 3 类与其相同，不多赘述。

表 3—8　左移位指令寻址方式一览表

指令格式	功　用	机器码	寻 址 方 式
ASL	A 左移一位	0A	累加器寻址
ASL $n	($n)左移一位	06 n	零页寻址
ASL $n,X	($n+X)左移一位	16 n	零页 X 变址
ASL $n2n1	($n2n1)左移位	0E n1 n2	绝对寻址
ASL $n2n1,X	($n2n1+X)左移	1E n1 n2	绝对 X 变址

1. 左移位

左移位指令 ASL 的功用是：将累加器或选定存储单元中的各位依次向左移一位，最高位(D7 位)移入进位 C 中，最低位(D0 位)补 0。由于 ASL 指令执行后相当于把移位前的数乘以 2，所以又称算术左移指令。ASL 指令将影响标志寄存器的 C、N、Z 标志。

如累加器 A 中的内容为 73，执行 ASL 一次后，A 中的内容变为 E6；若存储单元 $300 中的数据为 32，执行一次 ASL $300 后，$300 中的内容等于 64。显然，执行一次左移位指令，相当于完成了一次乘 2 操作。

2. 右移位

右移位指令 LSR 的功用是：将累加器或选定存储单元中的各位依次向右移一位，最低位(D0)移入进位 C 中，最高位(D7)补 0。

执行 LSR 指令对于无符号数和正数相当于除 2，但对于负数因最高位补 0，会把负数变成正数，所以 LSR 指令不能称为算术右移而称为逻辑右移。

LSR 指令也影响标志寄存器的 C、N、Z 位。

3. 循环左移位

循环左移指令 ROL 的功用是令累加器或存储单元中的数据各位连同进位 C 一起循环左移一位，使原来的最高位移入进位 C，原来的进位 C 移入最低位。

例 3—1 将一个二进制 16 位无符号数 37BDH 乘以 2(假设数据存放在 $300、$301 单元，高位字节存高位地址，低位字节存低位地址)。

解：只要用以下两条指令将 16 位无符号数整个左移一位就可实现题目的要求。指令为

ASL $300

ROL $301

两条指令的联合作用情况如图 3—2 所示。

由图可见，执行这两条指令后，$301、$300 单元的 16 位二进制数据依次向左移动了一位，相当于乘以 2。执行中，$300 单元左移位，其中数据的高位(D7 位)移入了进位 C，最低位(D0 位)补 0；$301 循环左移则把进位 C 中 $300 移来的进位移入了其最低位，从而完成了乘 2 的操作。指令执行后，$301、$300 单元存储的数据为

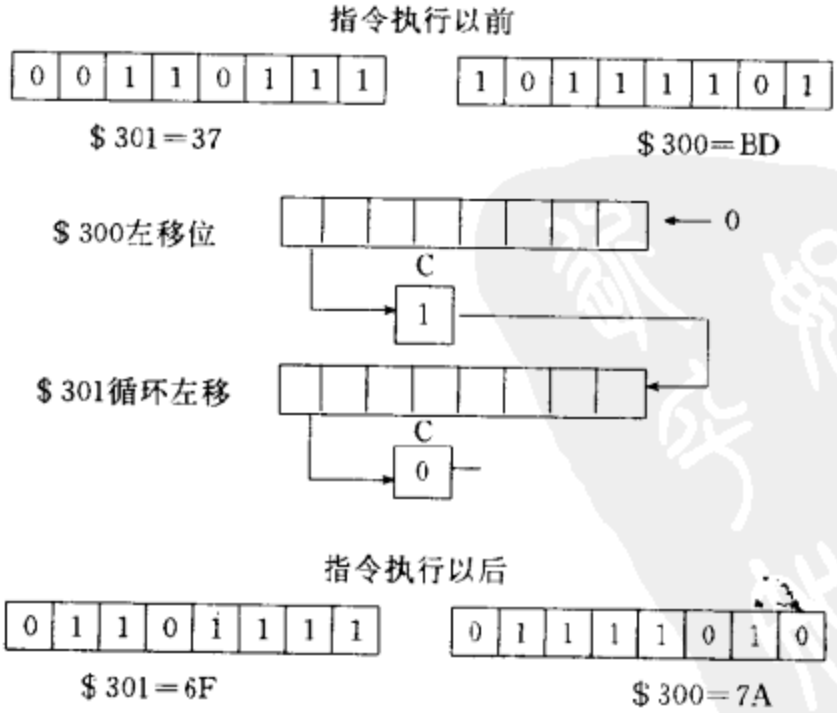


图 3—2　左循环乘 2 操作示意图

6F7AH＝37BDH×2

循环右移的情况与ROL指令相反，讨论从略。

显然，循环移位指令也将影响标志寄存器的C、N、Z位。

第二节　算术运算类指令

6527CPU内部的算术逻辑运算单元ALU，是完成各种算术、逻辑运算的基本部件，它可以实现加、减、与、或、异或、比较、移位、加1、减1等多项操作。它的操作对象有两个，一个是累加器A、一个是放在数据总线上的8位二进制数M。另外，由于变址寄存器X、Y可以完成加1、减1、比较等简单运算，所以数据、信息的传递将集中在A、X、Y、M中进行。运算过程中，运算结果与标志寄存器之间将发生相互影响，同时标志寄存器又是相对寻址跳转的重要依据，所以兼顾P寄存器的工作状态也是极其重要一环。

算术运算指令共有47条，分为：加、减、比较、加1、减1、标志位6类。

一、ADC——加法指令

ADC指令的功用是累加器与存储器或立即数进行带进位加法运算，结果仍存在累加器中。ADC指令的操作可表示为：

A＋M＋C→A

其中“M”代表某一存储器的内容或某一立即数，“C”为执行本指令前的P寄存器的进位状态。

ADC指令有8种寻址方式，其形式见表3—9。

表 3—9　ADC 指令寻址方式一览表

指令格式	功　用	机器码	寻 址 方 式
ADC # $n	A+# $n+C → A	69 n	立即寻址
ADC $n	A+($n)+C → A	65 n	零页寻址
ADC $n,X	A+($n+X)+C → A	75 n	零页 X 变址
ADC $n2n1	A+($n2n1)+C → A	6D n1 n2	绝对寻址
ADC $n2n1,X	A+($n2n1+X)+C→A	7D n1 n2	绝对 X 变址
ADC $n2n1,Y	A+($n2n1+Y)+C→A	79 n1 n2	绝对 Y 变址
ADC ($n,X)	A+(地址)+C → A	61 n	X 变间寻址
ADC ($n),Y	A+(地址)+C → A	71 n	Y 间变寻址

ADC 指令执行后，将影响标志寄存器的 N、V、Z、C 位。

ADC 是 6527 指令系统中的唯一的一条加法运算指令，且是带进位加法。因此，它特别适合多字节的加法运算。但在进行单字节的加法运算时，必须在 ADC 指令前预置进位 C＝0，才能保证结果不受进位的影响。

例 3—2 编写求两个单字节数 37H、28H 之和的子程序，并将和存入 $300 单元。

解：因将进行单字节加法运算，在加法指令 ADC 之前要加入清进位操作，程序见 No. 3—1（假定程序放在 $400 开始的内存中）。

No. 3—1 单字节加法子程序

```
0400 A9 37      LDA  # $37   ;取第一个立即数
0402 D8         CLD          ;预置禁止十进制运算标志
0403 18         CLC          ;预置进位 C=0
0404 69 28      ADC  # $28   ;A+28+C=37+28+0=5F→A
0406 8D 00 03   STA  $300    ;A=5F→ $300
0409 60         RTS          ;子程序返回
```

上述运算对标志位的影响可如下分析：因结果的最高位无进位则进位标志 C＝0；因结果的最高位等于 0 则符号位 N＝0；因结果不为 0 则零标志位 Z＝0；因结果没有超出单字节的取值范围则溢出位 V＝0。所以，标志寄存器的各位可表示为：

N	V	·	·	D	I	Z	C
0	0	·	·	0	·	0	0

例 3－3 设两个十六位二进制数 N1、N2 分别放在 $300～$303 单元中(如图 3－3 所示)，编写计算它们的和存放在 $304、$305 单元中(低位在前、高位在后)的子程序。

地址	数据	
300	N1L　35	N1 低位
301	N1H　67	N1 高位
302	N2L　A7	N2 低位
303	N2H　2D	N2 高位
304	N1L＋N2L	和的低位
305	N1H＋N2H	和的高位

图 3－3　两字节加法数据

实现程序见 N_O. 3－2(程序从 $400 单元开始存放)。

N_O. 3－2 两字节数加法子程序

```
0400 D8        CLD          ;清十进制运算标志
0401 18        CLC          ;清进位
0402 AD00 03   LDA  $300    ;($300) → A
0405 6D 02 03  ADC  $302    ;($300)＋($302)＋C → A
0408 8D 04 03  STA  $304    ;A → $304
040B AD01 03   LDA  $301    ;($301) → A
040E 6D 03 03  ADC  $303    ;($301)＋($303)＋C → A
```

```
0411 8D 05 03   STA   $305   ;A → $305
0414 60         RTS
```

程序 No. 3－2 中，第一步进行两数的低位字节相加(N1L＋N2L)，相加前应先将进位清零；第二步进行两数的高位字节相加，这时进位中将存放低位字节相加后的进位，因此在高位字节相加前不须清进位。显然，执行程序 No. 3－2 后，$304、$305 单元存放的和为 94DCH。

二、SBC——减法指令

SBC 指令的功用是：取累加器的数据减去存储器或立即数以及进位位的反码，结果仍存在累加器中。SBC 指令的操作可表示为：

$$A-M-\overline{C}\rightarrow A$$

其中“M”代表某一存储器的内容或某一立即数，“$\overline{C}$”为执行本指令前的 P 寄存器的进位位的反码。

SBC 指令也有 8 种寻址方式，其形式见表 3－10。

表 3－10　SBC 指令寻址方式一览表

指令格式	功　用	机器码	寻址方式
SBC # $n	A－# $n－* → A	E9 n	立即寻址
SBC $n	A－($n)－* → A	E5 n	零页寻址
SBC $n,X	A－($n+X)－* → A	F5 n	零页 X 变址
SBC $n2n1	A－($n2n1)－* → A	ED n1 n2	绝对寻址
SBC $n2n1,X	A－($n2n1+X)－*→A	FD n1 n2	绝对 X 变址
SBC $n2n1,Y	A－($n2n1+Y)－*→A	F9 n1 n2	绝对 Y 变址
SBC ($n,X)	A－(地址)－* → A	E1 n	X 变间寻址
SBC ($n),Y	A－(地址)－* → A	F1 n	Y 间变寻址

注——表中的“*”号表示进位位 C 的反码。

SBC 指令执行后，将影响标志寄存器的 N、V、Z、C 位。若 C＝1 时说明没有借位；C＝0 说明有借位。

SBC 是 6527 指令系统中的唯一的一条减法运算指令，且是带进位减法。因此，它特别适合多字节的减法运算。但在进行单字节的减法运算时，必须在 SBC 指令前预置进位 C＝1，才能保证结果不受进位的影响。

例 3－4 设累加器 A 中的内容为 08，$1F 单元的数据为 09，编写求

A－($1F)＝?

的子程序(程序从 $400 开始存放)。

解：这是两个单字节数的减法运算，必须先预置 $\overline{C}=0$ (即 C＝1)，程序见 No. 3－3。

No. 3－3 单字节减法子程序

```
0400  D8     CLD
0401  38     SEC            ;预置 C̄=1
0402  A9  08LDA   #$08  ;
0404  E5  1FSBC   $1F     ;A－($1F)－C̄→A
0406  60     RTS
```

程序 No. 3－3 执行后，累加器 A 中的内容为 FFH (11111111B，即－1 的补码)。运算结果对标志位的影响是：因结果的最高位等于 1，则符号位 N＝1；因结果非零，则零标志 Z＝0；因运算中有借位，则 C＝0；因无溢出则 V＝0。因此上述运算对标志寄存器的影响可表示为：

N	V	·	·	D	I	Z	C
1	0	·	·	0	·	0	0

三、比较指令

1. CMP——累加器 A 与存储器内容或立即数比较

累加器与存储器或立即数比较的操作是：A－M(M 是立即数或存储器中的数据)，但结果不送累加器，指令执行后仅影响标志寄存器的 N、Z、C 标志位。

CMP 指令的操作虽然执行减法运算，但它与减法指令有两点区别：第一、进位标志 C 不参加运算，因此在使用 CMP 指令之前不须对进位位操作；第二、运算结果不送入累加器，因而执行 CMP 指令后不改变 A 中的内容。

CMP 指令常用来判断累加器 A 和 M 是否相等或比较大小。执行 CMP 指令后，标志位 N、Z、C 将发生变化，如 Z＝1 则表示 A＝M；若 C＝1 则表示 A 够减、无借位，A≥M；若 C＝0 则表示不够减、有借位，A＜M。

CMP 指令与 SBC 一样，也有 8 种相同的寻址方式，见表 3－11。

表 3－11　CMP 指令寻址方式一览表

指令格式	功　用	机器码	寻址方式
CMP ＃$n	A－＃$n	C9 n	立即寻址
CMP $n	A－($n)	C5 n	零页寻址
CMP $n,X	A－($n+X)	D5 n	零页 X 变址
CMP $n2n1	A－($n2n1)	CD n1 n2	绝对寻址
CMP $n2n1,X	A－($n2n1+X)	DD n1 n2	绝对 X 变址
CMP $n2n1,Y	A－($n2n1+Y)	D9 n1 n2	绝对 Y 变址
CMP ($n,X)	A－(地址)	C1 n	X 变间寻址
CMP ($n),Y	A－(地址)	D1 n	Y 间变寻址

2. CPX——变址寄存器 X 与存储器内容或立即数比较

X 与存储器内容或立即数比较的操作是：X－M。其功能与 CMP 指令相似，只不过是把累加器 A 换成变址寄存器 X。

CPX 的寻址方式只有 3 种(见表 3－12)，影响标志寄存器的情况也是 N、Z、C 位。

3. CPY——变址寄存器 Y 与存储器内容或立即数比较

Y 与存储器内容或立即数比较的操作是：Y－M。其功能与 CPX 指令相似，只不过是把 X 换成 Y。CPY 的寻址方式也只有 3 种(见表 3－12)，影响标志寄存器的情况也是 N、Z、C 位。

表 3－12　CPX、CPY 指令寻址方式一览表

指令格式	功　用	机器码	寻址方式
CPX #$n	X－#$n	E0 n	立即寻址
CPX $n	X－($n)	E4 n	零页寻址
CPX $n2n1	X－($n2n1)	EC n1 n2	绝对寻址
CPY #$n	Y－#$n	C0 n	立即寻址
CPY $n	Y－($n)	C4 n	零页寻址
CPY $n2n1	Y－($n2n1)	CC n1 n2	绝对寻址

比较指令常与条件转移指令配合使用，以便实现程序的条件转移，故它在分支程序的设计中占有极重要的地位。

四、加 1 指令

INC 指令也叫增量指令，它的功用是令存储器单元的内容加 1，即 M＋1→M(M 为存储器中的数据)。INC 指令有 4 种寻址方式(见表 3－13)，执行后将影响标志寄存器的 N、Z 标志位。

INX、INY 为隐含寻址方式的单字节指令，它们分别控制变址寄存器 X、Y 加 1，执行后影响 N、Z 标志位。

五、DEC——减 1 指令

DEC 指令也叫减量指令，它的功用是令存储器单元的内容减 1，即 M－1→M(M 为存储器中的数据)。DEC 指令也有 4 种寻址方式(见表 3－13)，执行后将影响标志寄存器的 N、Z 标志位。

DEX、DEY 为隐含寻址方式的单字节指令，它们分别控制变址寄存器 X、Y 减 1，执行后影响 N、Z 标志位。

表 3－13　INC、DEC 指令寻址方式一览表

指令格式	功　用	机器码	寻址方式
INC $n	($n)+1	E6 n	零页寻址
INC $n,X	($n+X)+1	F6 n	零页 X 变址
INC $n2n1	($n2n1)+1	EE n1 n2	绝对寻址
INC $n2n1,X	($n2n1－X)+1	FE n1 n2	绝对 X 变址
DEC $n	($n)－1	C6 n	零页寻址
DEC $n,X	($n+X)－1	D6 n	零页 X 变址
DEC $n2n1	($n2n1)－1	CE n1 n2	绝对寻址
DEC $n2n1,X	($n2n1+X)－1	DE n1 n2	绝对 X 变址

六、标志位指令

标志寄存器的各标志位虽然与条件转移指令密切相关，但也可以通过专用指令进行预置。

标志位指令均为隐含寻址的单字节指令，它们是：

SEC——将进位标志位置 1，即 1→C，操作码＝38。

SED——将十进制运算标志位置 1，即 1→D，操作码＝F8。

CLC——将进位标志位置 0，即 0→C，操作码＝18。

CLD——将十进制运算标志位置 0，即 0→D，操作码＝D8。

CLV——将溢出标志位置 0，即 0→V，操作码＝B8。6527 没有置 V 等于 1 的指令，可通过 BIT 指令来影响标志位而置 V＝1。

第三节　逻辑运算与位测试类指令

逻辑运算指令共有 3 类、24 条。它们有三个共同的特点：第一、逻辑运算操作按位独立进行，彼此之间不存在进位、借位关系；第二、均有 8 种寻址方式；第三、执行指令后只影响标志寄存器的 N、Z 位。

位测试指令只有两条，它是通过累加器 A 对寄存器的各位进行检测的。

一、逻辑运算指令

1. AND——累加器与存储器内容逻辑“与”运算

AND 指令的功用是将累加器 A 与存储器中的内容（或立即数）按位进行逻辑“与”运算，运算后的结果送入累加器，而存储器中的内容不变。可表示为 A∧M→A（“∧”表示“与”逻辑运算；这里的 M 既可以是存储器的内容，也可以是一个立即数）。AND 有 8 种寻址方式，见表 3－14。

AND 指令常用于屏蔽数据的某些位或者取出数据的某些位，以组成一个新的数据。

表 3－14　AND 指令寻址方式一览表

指令格式	功　用	机器码	寻址方式
AND # $n	A∧# $n → A	29 n	立即寻址
AND $n	A∧($n) → A	25 n	零页寻址
AND $n,X	A∧($n+X) → A	35 n	零页 X 变址
AND $n2n1	A∧($n2n1) → A	2D n1 n2	绝对寻址
AND $n2n1,X	A∧($n2n1+X)→A	3D n1 n2	绝对 X 变址
AND $n2n1,Y	A∧($n2n1+Y)→A	39 n1 n2	绝对 Y 变址
AND ($n,X)	A∧(地址) → A	21 n	X 变间寻址
AND ($n),Y	A∧(地址) → A	31 n	Y 间变寻址

例 3－5 已知 $300 单元的数据为 7D。要求取出 $300 的低四位置入 $301 单元的低四位、$301 单元的高四位清零；取出 $300 单元的高四位置入 $302 单元的高四位，$302 单元的低四位清零。

解：可使用 AND 指令，先取 $300 单元的数据，把高四位屏蔽后，置入 $301 单元；再取 $300 单元的数据屏蔽其低四位后，置入 $302 单元。实现程序见 No. 3－4（程序从 $400 开始存放）。

No. 3－4 数据分割子程序

```
0400 AD 00 03   LDA  $300    ;取 $300 单元的数据置入 A
0403 48         PHA          ;A 中的数值压栈
0404 29 0F      AND  # $0F   ;7D AND OF=0D
0406 8D 01 03   STA  $301    ;0D→ $301
0409 68         PLA          ;A 值出栈，A=7D
040A 29 F0      AND  # $F0   ;7D AND FO=70
```

```
040C 8D 02 03   STA $302   ;70→$302
040F 60         RTS
```

2. ORA——累加器与存储器内容逻辑"或"运算

ORA 指令的功用是将累加器 A 与存储器中的内容按位进行逻辑"或"运算，运算后的结果送入累加器，而存储器中的内容不变。可表示为 A∨M→A(M 的意义同上，"∨"表示"或"逻辑运算)。ORA 也有 8 种寻址方式，见表 3－15。

表 3－15　ORA 指令寻址方式一览表

指令格式	功　用	机器码	寻址方式
ORA #$n	A∨#$n→A	09 n	立即寻址
ORA $n	A∨($n)→A	05 n	零页寻址
ORA $n,X	A∨($n+X)→A	15 n	零页 X 变址
ORA $n2n1	A∨($n2n1)→A	0D n1 n2	绝对寻址
ORA $n2n1,X	A∨($n2n1+X)→A	1D n1 n2	绝对 X 变址
ORA $n2n1,Y	A∨($n2n1+Y)→A	19 n1 n2	绝对 Y 变址
ORA ($n,X)	A∨(地址)→A	01 n	X 变间寻址
ORA ($n),Y	A∨(地址)→A	11 n	Y 间变寻址

ORA 指令常用于使某些位置 1 或者求混合信息。例如执行"ORA #$80"会无条件地把 A 的最高位置 1。如欲将 $300 单元的高四位置 1、低四位不变，可执行

```
LDA $300
ORA #$F0
STA $300
```

三条指令。

3. EOR——累加器与存储器内容逻辑"异或"运算

EOR 指令的功用是将累加器 A 与存储器中的内容按位进行逻辑"异或"运算，运算后的结果送入累加器，而存储器

中的内容不变。可表示为 A ⊙ M→A(M 的意义同上，“⊙”表示逻辑异或运算)。EOR 也有 8 种寻址方式，见表 3—16。

表 3—16　EOR 指令寻址方式一览表

指令格式	功　用	机器码	寻址方式
EOR # $n	A⊙# $n → A	49 n	立即寻址
EOR $n	A⊙($n) → A	45 n	零页寻址
EOR $n,X	A⊙($n+X) → A	55 n	零页 X 变址
EOR $n2n1	A⊙($n2n1) → A	4D n1 n2	绝对寻址
EOR $n2n1,X	A⊙($n2n1+X)→A	5D n1 n2	绝对 X 变址
EOR $n2n1,Y	A⊙($n2n1+Y)→A	59 n1 n2	绝对 Y 变址
EOR ($n,X)	A⊙(地址) → A	41 n	X 变间寻址
EOR ($n),Y	A⊙(地址) → A	51 n	Y 间变寻址

EOR 指令常用于求反码。例如执行“EOR # $FF”可将累加器 A 中的内容按位取反，这是因为逻辑“异或”运算的性质是“两数相同、结果为 0；两数不同、结果为 1”，故当各位都异或 1 时就达到了取反的目的。因 6527 没有求反码的指令，所以常用“EOR”指令间接求得反码。

二、位测试指令

BIT——用累加器检测存储器各位

BIT 指令的操作分以下四个步骤：

1. 取累加器 A 中的内容与指定存储单元中的内容按位相“与”。

2. 若 A∧M=0，则置标志寄存器的 Z=1；否则置 Z=0。

3. 把存储器内容的 D7 位(最高位)送入标志寄存器的符号标志位 N。

4. 把存储器内容的 D6 位(次高位)送入标志寄存器的溢出标志位 V。

注意,BIT 指令执行后,累加器 A、存储器中的内容均不改变。

BIT 指令只有两种寻址方式,见表 3－17。

表 3－17　BIT 指令寻址方式一览表

指令格式	功　用	机器码	寻址方式
BIT $n	A∧($n)	24 n	零页寻址
BIT $n2n1	A∧($n2n1)	2C n1 n2	绝对寻址

例 3－6 已知累加器 A 中的数据为“A6”，$300 单元的内容为“E0”，判断执行“BIT $300”指令后标志寄存器的情况。

解:因 $300 单元的 D7 位等于 1，则 P 寄存器的 N＝1；因 $300 单元的 D6 位等于 1,则 P 的 V＝1；因 A6 ∧E0≠0，则 P 的 Z＝0。所以执行上述指令后，标志寄存器的情况如下：

N	V	·	·	D	I	Z	C
1	1	·	·	·	·	0	·

第四节　程序控制类指令

程序控制类指令包括程序转移指令和 CPU 控制指令两大部分。

它们的共同特点是:改变程序的顺序执行次序，使程序计数器 PC 不再按照不断自动加 1 的操作向下运行，而是直接跳转到新的地址去继续执行程序。

程序转移类指令的设置体现了分支程序设计的需求。程序的设计技巧，往往就由灵活、巧妙地使用这些指令而显现出来。

程序控制类指令共有 16 条，分条件转移、无条件转移、子程序及返回、中断和空操作 5 类。

一、JMP——无条件转移指令

因 JMP 指令执行时，并不理会标志寄存器中的各项条件而直接转移，所以称为无条件转移指令。

JMP 只有两种寻址方式，见表 3—18。

表 3—18　JMP 指令寻址方式一览表

指令格式	功　用	机器码	寻址方式
JMP $n2n1	$n2n1 → PC	4C n1 n2	绝对寻址
JMP ($n2n1)	($n2n1+1, $n2n1)→PC	6C n1 n2	间接寻址

JMP 指令常在需要转移程序的执行路线时使用。它的执行操作如下：

对于绝对寻址方式，CPU 把“n2”送入 PCH、“n1 ”送入 PCL，然后转向“$n2n1”地址执行程序。

对于间接寻址方式，CPU 将把($n2n1+1)单元的数据送入 PCH，把($n2n1)单元的数据送入 PCL，然后转向 PCH、PCL 所指向的地址执行程序。

二、条件转移指令

条件转移指令共有 8 条，见表 3—19。

表 3—19　条件转移指令寻址方式一览表

指令格式	功　用	机器码	寻址方式
BCC $n	C=0 转移	90 n	相对寻址
BCS $n	C=1 转移	B0 n	〃
BEQ $n	结果为零转移(Z=1)	F0 n	〃
BNE $n	结果非零转移(Z=0)	D0 n	〃
BVC $n	无溢出转移(V=0)	50 n	〃
BVS $n	有溢出转移(V=1)	70 n	〃
BMI $n	结果为负转移(N=1)	30 n	〃
BPL $n	结果为正转移(N=0)	10 n	〃

条件转移指令具有以下四个共同特点：

第一、它们的操作实质就是对指令计数器 PC 重新赋值。

第二、它们的操作都是根据标志寄存器中的某一状态值进行的。

第三、它们都采用相对寻址方式，均为两字节指令(前一字节是操作码、后一字节是偏移量)。

第四、它们执行与否，实际上都是根据前一条指令的执行结果而决定程序走向的，指令执行后都不改变标志寄存器的状态。

三、子程序指令

子程序指令只有两条：转子指令和返主指令。

1. JSR——转子指令

该指令执行后，CPU 转到该指令操作数所指出的地址去执行子程序。它只有绝对寻址一种方式，指令格式为：

JSR $n2n1

指令执行的第一步是把 JSR $n2n1 指令的第三字节在内存中的地址压栈保存，以备执行到子程序返回指令时取出；第二步把“n2n1”送入 PC，再根据 PC 指出的地址转移到子程序的入口，执行子程序。

转子指令也是无条件转移指令，它与 JMP 指令的区别为：JMP 指令执行时不保存断点地址，控制程序转出后不再返回；而 JSR 指令执行时，要压栈保存断点地址，子程序执行完毕时还要返回主程序。

2. RTS——返主指令

执行该指令后，CPU 将返回到主程序的断点处。它是采取隐含寻址方式的单字节指令，通常放在子程序的末尾，以便在子程序执行完毕后返回主程序。

RTS 指令的执行过程是：从堆栈中取出栈顶两个单元的内容，它们就是转子时压栈的 JSR 指令第三字节在内存的地址。将此地址送入 PC 后，经 PC+1 修正后得到返回地址，即 JSR 指令下面指令的首地址。CPU 将根据 PC 的内容，返回主程序继续执行。

四、中断指令

中断指令共有 3 条，见表 3－20。

表 3－20　中断指令寻址方式一览表

指令格式	功　用	机器码	寻址方式
CLI	开中断，0→I	58	隐含寻址
SEI	关中断，1→I	78	〃
RTI	中断返回	40	〃

CLI、SEI 是可屏蔽中断控制指令。当标志寄存器的 I＝0 时，可屏蔽中断允许，6527 响应中断后，将转到可屏蔽中断

服务子程序执行程序。当标志寄存器的 I＝1 时，可屏蔽中断处于关断状态，这时 6527 不响应可屏蔽中断申请。

RTI 为中断程序返回指令。当 6527 响应中断后，将把断点地址和标志寄存器 P 的内容压栈保存，然后转到中断服务程序执行。当中断服务程序执行完毕后，将通过设置在中断程序最后的 RTI 指令返回。返回操作是，从堆栈弹出断点地址和 P 值，恢复中断服务前的现场，继续执行主程序。

五、空操作指令

空操作 NOP 也是隐含寻址方式的单字节指令。CPU 执行 NOP 指令，除了使 PC＋1→PC、占用两个时钟周期的机时外，不做其它任何操作，因此称为空操作。

空操作指令常在程序的调试中使用。如在调试含有子程序调用的程序中，可用 NOP 代替 JSR 所占用的字节，先调试主程序，待主程序调通后，再替换 JSR 的指令代码。这样，可避免程序调试阶段因地址变化而造成的混乱。

56 条基本指令加上不同的寻址方式后，组成了 150 条指令，由它们构成了 6527 机器语言的指令系统。各指令的格式、功能、寻址方式、机器码、对标志寄存器各位的影响情况以及指令执行周期等情况，请参阅附录。

第四章　汇编与 BASIC 的联姻

由于目前上市的多数电脑学习机没有配置汇编语言工作系统，给编写和运行机器语言程序带来了困难。但欲提高编程效率、提高程序的运行速度，编写出结构好、效率高、功能强的的程序又离不开机器语言。这是一个大难题，怎么解决呢？

唯一的途径就是：BASIC 语言与汇编语言联姻，使用 BASIC 语言的 POKE、PEEK、CALL 指令，在 BASIC 语言环境下，编写、运行机器语言程序。

本章将重点讨论在 BASIC 语言环境下，如何汇编、阅读机器语言程序，如何编写、调试机器语言程序，如何运行、修改机器语言程序。并给出有效的工具程序以方便上述操作。

本章介绍的汇编程序，目的在于提供操作工具，读者不必强求完全看懂，只要会输入使用就可以了。

第一节　机器语言程序的写入与运行

BS・2A 版本的 F BASIC 工作系统（有的学习机误为 G BASIC），是最经典、最严谨的 BASIC 工作系统。在该系统下可以流畅地编写各种游戏程序和英文提示的应用程序，我们仅以该系统为例，讨论机器语言程序写入内存的方法，其一般原理当然也适用于其它 BASIC 系统。

一、机器语言程序存储地址的确定

在 F BASIC 工作状态下写入机器语言程序，必须处理好两个系统程序的结合、兼容问题。处理不好轻则程序不能运行，重则造成死机，使辛苦编写、输入的程序毁于一旦，那将是令人十分沮丧的事。因此，编程前必须清楚机器语言程序放置的地址范围。

1. RAM 区

6527CPU 虽然直接寻址能力为 64K 字节，但并不是任何一段地址都可以供用户使用。由表 1－6“6527CPU 管理的内存地址分布表”可知，RAM 区只有 $00～$7FF 一段的 2K 和 $6000～$7FFF 一段的 8K 字节。其它空间有的被系统占用，有的没有硬件环境支持，均不能使用。因此，机器语言程序的存储空间只能在 $00～$7FF、$6000～$7FFF 两段 RAM 中寻找。

2. 系统 RAM 区

对于 $00～$7FF 的 RAM 段，由于被 F BASIC 系统工作广泛使用，故称为系统 RAM 区。该区的大部分单元被系统使用，剩余单元很零碎，已基本上不能用于存放程序。因此，我们的希望应在后面的一块 RAM 中。

3. 用户 RAM 区

BS·2A 版本的 F BASIC 系统，规定用户的 BASIC 程序存储空间从 $703E 开始，内存地址为 $703E～$7FFF 共 4030 个内存单元。因此，当 BASIC 程序量较小时，可以考虑使用这一空间。通常，为避免机器语言程序与 FBASIC 程序的冲突，一般把机器语言程序尽量往后放，具体存放地址可用倒推法算出。如机器语言程序长度为 500 个字节，换算为

十六进制数为1F4H，用7FFFH减1F4H，其差等于7E0CH。故可选存放机器语言程序的首址为$7E00。

但当BASIC的程序量较大、或者程序中开辟了较多的数组时，程序运行中，因变量区的扩大可能会冲掉放在后面的机器语言程序。因此，到BASIC的RAM区中为机器语言程序争地盘，虽然可行但不可靠。

4. F BASIC空闲区

实际上，在F BASIC程序的存储区之前，还有一大段RAM区，就是地址为$6000～$6FFF的一段4K空间。这段RAM一般在浮点BASIC、中文BASIC或LOGO、文字处理等系统中使用，在F BASIC系统中是一块空闲地。但闲地不闲，许多机型使用这一段作为各系统切换的RAM驻留程序区。当把机器语言程序写入这一区域时，一旦退出F BASIC工作系统后，再重新进入时，切换程序可能要冲掉部分机器语言程序。因此，使用这段RAM区必须根据自己的机型，找出实际的空区。

寻找空区的方法是：

(1) 开机后，进入F BASIC工作系统，编写程序N_0. 4—1。

```
5 REM N0. 4—1
10 FOR I=&6000 TO &6FFF
20 POKE I,&77
30 NEXT
```

(2)运行上述程序，将把$6000～$6FFF共4K单元全部置为77H(选择77H，是为了对各RAM单元置入特殊标记，可选择任意方便识别的数字)。

(3) 按复位键或选择退出项，返回主菜单画面后，再进入

F BASIC 工作系统，键入程序 No. 4－2。

```
5 REM No. 4－2
10 FOR I=&6000 TO &6FFF
20 PRINT HEX$(I);"=";HEX$(PEEK(I)),
30 NEXT
```

（4）运行程序 No. 4－2，将分栏打印出 $6000～ $6FFF 单元中的内容。一般来说，凡内容为置入标志数据的单元，均为空区。

5. 扩展 RAM 区

有的机型，如裕兴 YX4・0 以上版本的电脑学习机，均配有 32K 用户 RAM。这 32KRAM 被分为 4 段，每段 8K，段号依次为 0～3，映射地址均为 $6000～ $7FFF。F BASIC 程序仅使用 RAM 区的第 0 段，尚有 24K 空间供机器语言程序使用，这是极大的内存资源。对扩展 RAM 区的使用，要进行 RAM 切段操作，切段地址为 $5000，切段方法是把 RAM 段号置入 $5000 单元。

使用扩展 RAM 时，要注意处理好，RAM 段间的衔接。

二、使用 POKE 语句写入机器语言程序的方法

普通 BASIC 语言中的 POKE 语句，一次只能向指定地址单元写入一个数据，而 F BASIC 是一种增强型的高级语言，它的 POKE 语句一次可向指定地址写入多个数据，这也是 F BASIC 语言的又一特点。下面通过画面右移子程序 No. 4－3 介绍这一方法的应用。

No. 4－3 画面右移子程序

```
START    PHA              ;A 压栈保护现场
         LDX  #$00        ;0→X
```

```
        LDY   # $00      ;0→Y
MOVE    STX   $2005      ;X→$2005
        STY   $2005      ;Y→$2005
        LDA   # $FF
YANSHI  STA   $F3        ;以$F3为延时计数器
        JSR   $86B9      ;调STOP键扫描子程序作为短延时
        DEC   $F3        ;延时计数器减1
        BNE   YANSHI     ;延时没结束则继续
        INX              ;X+1→X
        BNE   MOVE       ;X≠0继续移位
        PLA              ;X=0,移位结束,A值出栈,
        RTS              恢复现场,返回主程序
```

如果选定程序的存放地址为$6FE0,则No.4—3对应的6527机器语言子程序如No.4—4所示。

No.4—4 画面右移子程序

```
6FE0  48                    PHA
6FE1  A2  00          LDX   # $00
6FE3  A0  00          LDY   # $00
6FE5  8E  05  20      STX   $2005
6FE8  8C  05  20      STY   $2005
6FEB  A9  FF          LDA   $FF
6FED  85  F3          STA   $F3
6FEF  20  B9  86      JSR   $86B9
6FF2  C6  F3          DEC   $F3
6FF4  D0  F9          BNE   $6EED
6FF6  E8              INX
6FF7  D0  EC          BNE   $6EE5
6FF9  68              PLA
6FFA  60              RTS
```

把程序 No. 4－4 写入内存的 F BASIC 程序为 No. 4－5。

No. 4－5 把画面右移子程序写入内存的 BASIC 程序

POKE &6FE0,&48,&A2,0,&A0,0,&8E,5,&20,&8C,5,&20,&A9,&FF,&85,&F3,&20,&B9,&86,&C6,&F3,&D0,&F9,&E8,&D0,&EC,&68,&60

No. 4－5 中，所有十六进制数的前缀“&H”均省略了“H”，这是 F BASIC 允许的一种简写形式。

由以上讨论可见，一行 POKE 语句最多可一次写入 60 个数据，当编写暂缓执行型程序时则最多可写 48 个数据（由程序列表时每行 256 个字符的限制而定）。

三、使用循环语句写入机器语言程序的方法

用循环语句把程序 No. 4－4 写入内存的 F BASIC 程序见 No. 4－6。

```
5 REM No. 4－6 把画面右移子程序写入内存
10 FOR I=0 TO 26
20 READ A : POKE &6FE0 + I, A
30 NEXT
40 DATA &48, &A2, 0, &A0, 0, &8E, 5, &20, &8C, 5, &20,&A9,
   &FF, &85,&F3, &20, &B9, &86, &C6, &F3, &D0,&F9, &E8,
   &D0, &EC, &68, &60
```

四、机器语言程序写入内存程序的优化

通过使用 No. 4－5、No. 4－6 把机器语言程序写入内存的实践，都会产生一种别扭的感觉，原因是：每个数据前面都要带一个前缀 "&" ，它一方面影响了程序的输入速度；另一方面使机器语言程序变得不直观。若把十六进制数改为十进

制数，虽然可以提高程序的输入速度，但将使机器语言程序变得更加难读，不利程序的修改。

那么，是否有两全其美的方法呢？

程序 No. 4－7 较好的解决了这一问题。

```
5 REM No. 4－7 把画面右移子程序写入内存
10 FOR I=0 TO 26
20 READ A$ : POKE &6FE0 + I, VAL("&" +A$)
30 NEXT
40 DATA 48, A2, 0, A0, 0, 8E, 5, 20, 8C, 5, 20, A9,FF, 85,
   F3, 20,B9, 86, C6, F3, D0, F9, E8, D0, EC,68, 60
```

程序 No. 4－7 中，先把数据当作字符串从数据区读出，然后再把字符转化为数据写入内存，从而解决了数据输入方便与数据读、改直观的矛盾。因此，程序 No. 4－7 是最优化的机器语言写入方法，也是最常用的机器语言写入方法。一般来说，当程序量较小时使用 No. 4－5；当程序量较大时多使用 No. 4－7。

五、机器语言程序的运行

F BASIC 语言中的 CALL 指令，可以完成机器语言程序的运行操作，其语句格式为：

CALL XXXX（机器语言子程序首地址）

现在我们可以进行画面右移操作了。

在程序 No. 4－5(或 No. 4－6、No. 4－7)运行一次后，即可把程序删去。只要键入

CALL &6FE0

即可令画面连续右移。

改变 $F3 的初值可调整画面的右移速度；

改变 X 寄存器的初值可以控制画面右移的距离；

改“INX”为“DEX”，可令画面左移；

改“INX”为“INY”或“DEY”，可令画面上、下移动。

因此，程序 No. 4－3 是一个画面位移的通用程序。

第二节　6527 反汇编

为方便读者阅读内存 ROM 中的系统程序，学习编程技巧，现提供一个用 F BASIC 语言编写的 6527 反汇编程序，程序清单见 No. 4－8，本程序适用于任何版本的 F BASIC 语言系统。

在输入程序 No. 4－8 的数据语句行时，要注意每一个助记符中的空格。DATA 语句中的数据是按助记符的字母顺序装入的，如欲加快反汇编的运行速度，可调整 DATA 语句中的数据次序，把最常用的指令靠前放置。注意——DATA 中的数据每三个为一组，第一个为指令代码；第二个为汇编指令格式；第三个为指令的字节数。

```
5 REM No. 4－8 6527 反汇编
10 CLS: INPUT " 输入反汇编首地址 &H"; N$:
   N= VAL( "&" + N$)
20 PRINT : DD=N : GOSUB 2000
30 AA= PEEK (N) : I=0 : RESTORE
40 SS= N : GOSUB 1000 : N= N + 1
50 READ A$, B$, C : A= VAL ("&" + A$) :
   IF AA=A THEN 80
60 I=I + 1 : IF I < 151 THEN 50
70 B$ = "???" : C=1
80 IF C=1 THEN POKE 72, 15 : PRINT B$ ; : GOTO 200
```

```
90 SS=N : GOSUB 1000 : N=N + 1
100 IF C= 2 THEN 120
110 SS= N : GOSUB 1000 : N= N + 1
120 POKE 72, 15
130 FOR K=1 TO LEN (B$) : D$ = MID$ (B$, K, 1)
140 IF D$ = "3" THEN GOSUB 1100
150 IF D$ = "2" THEN SS= N - 1 : GOSUB 1000 : GOTO 190
160 IF D$ = "1" THEN SS= N - 1 : GOSUB 1000 : GOTO 190
170 IF D$ = "/" THEN D$ = ","
180 PRINT D$ ;
190 NEXT
200 H= H + 1 : IF H < 20 THEN 20
210 PAUSE : H= 0 : GOTO 20
1000 SS$ = HEX$ ( PEEK( SS ))
1010 IF LEN( SS$ )= 1 THEN SS$ = "0" + SS$
1020 PRINT SS$ ; : RETURN
1100 S= VAL( "&" + SS$ ) : IF S < 128 THEN E= N + S : GO-
     TO 1120
1110 E= N - 256 + S
1120 DD= E : GOSUB 2000
1130 RETURN 190
2000 DD$ = HEX$ ( DD )
2010 IF LEN( DD$ ) < 4 THEN DD$ = "0" + DD$ :
     GOTO 2010
2020 PRINT DD$ " " ; : RETURN
220 DATA 69,ADC # $2,2,6D,ADC $21,3,65,ADC $2,2,61,
    ADC ($2/X),2,71,ADC ($2)/Y,2,75,ADC $2/X,2,7D,
    ADC $21/X,3,79,ADC $21/Y,3,29,AND # $2,2,2D,AND
    $21,3,25,AND $2,2,21,AND ($2/X),2
230 DATA 31,AND ($2)/Y,2,35,AND $2/X,2,3D,AND $21/X,
```

```
3,39,AND $21/Y,3,E,ASL $21,3,6,ASL $2,2,A,ASL,1,
16,ASL $2/X,2,1E,ASL $21/X,3,90,BCC $3,2,B0,BCS 3,
2,F0,BEQ $3,2,D0,BNE $3,2,50,BVC $3,2
240 DATA 70,BVS 3,2,30,BMI $3,2,10,BPL $3,2,24,BIT $2,
2,2C,BIT $21,3,0,BRK,1,18,CLC,1,D8,CLD,1,58,C LI,1,
B8,CLV,1,C9,CMP # $2,2,CD,CMP $21,3,C1,CMP ( $2/
X),2,D1,CMP ($2)/Y,2,D5,CMP $2/X,2,
250 DATA DD,CMP $21/X,3,D9,CMP $21/Y,3,C5,CMP $2,2,
E0,CPX # $2,2,E4,CPX $2,2,EC,CPX $21,3,C0,CPY # $
2,2,C4,CPY $2,2,CC,CPY $21,3,C6,DEC $2,2,D6,DEC 2/
X,2,CE,DEC $21,3,DE,DEC $21/X,3
260 DATA CA,DEX,1,88,DEY,1,49,EOR # $2,2,45,EOR $2,2,
55,EOR $2/X,2,4D,EOR $21,3,5D,EOR $21/X,3,59,EOR
$21/Y,3,41,EOR ($2/X),2,51,EOR ($2)/Y,2,E6,INC
$2, 2,F6,INC $2/X,2,EE,INC $21,3
270 DATA FE,INC $21/X,3,E8,INX,1,C8,INY,1,4C,JMP $21,
3,6C,JMP ($21),3,20,JSR $21,3,A9,LDA # $2,2,A5,LDA
$2,2,B5,LDA $2/X,2,AD,LDA $21,3,BD,LDA $21/X,3,
B9, LDA $21/Y,3,A1,LDA ($2/X),2,B1,LDA ($2)/Y,2
280 DATA A2,LDX # $2,2,A6,LDX $2,2,B6,LDX $2/Y,2,AE,
LDX $21,3,BE,LDX $21/Y,3,A0,LDY # $2,2,A4,LDY
$2,2,B4, LDY $2/X,2,AC,LDY $21,3,BC,LDY $21/X,3,
4A,LSR,1,46,LSR $2,2,56,LSR $2/X,2,4E,LSR $21,3
290 DATA 5E,LSR $21/X,3,EA,NoP,1,9,ORA # $2,2,5,ORA
$2,2,15,ORA $2/X,2,D,ORA $21,3,1D,ORA $21/X,3,
19,ORA $21/Y,3,1,ORA ($2/X),2,11,ORA ($2)/Y,2,48,
PHA,1,8,PHP,1,68,PLA,1,28,PLP,1,2A,ROL,1
300 DATA 26,ROL $2,2,36,ROL $2/X,2,2E,ROL $21,3,3E,
ROL $21/X,3,6A,ROR,1,66,ROR $2,2,76,ROR $2/X,2,
6E,ROR $21,3,7E,ROR $21/X,3,40,RTI,1,60,RTS,1,E9,
```

```
    SBC #$2,2,E5,SBC $2,2,F5,SBC $2/X,2
310 DATA ED,SBC $21,3,FD,SBC $21/X,3,F9,SBC $21/Y,3,
    E1,SBC ($2/X),2,F1,SBC ($2)/Y,2,38,SEC,1,F8,SED,1,
    78,SEI,1,85,STA $2,2,95,STA $2/X,2,8D,STA $21,3,
    9D,STA $21/X,3,99,STA $21/Y,3
320 DATA 81,STA ($2/X),2,91,STA ($2)/Y,2,86,STX $2,2,
    96,STX $2/Y,2,8E,STX $21,3,84,STY $2,2,94,STY $2/
    X,2,8C,STY $21,3,AA,TAX,1,A8,TAY,1,BA,TSX,1,8A,
    TXA,1,9A,TXS,1,98,TYA,1
```

程序输入完毕，检查无误后运行程序，根据提示，直接输入十六进制的反汇编起始地址，回车后即开始反汇编。每反汇编二十行自动暂停，按任一键后继续反汇编。按 STOP 键后停止反汇编。

程序 No.4－8 可反汇编出任一起始地址的内存程序，其反汇编格式与中华学习机的小汇编完全相同。

第三节　6527 小监控系统

本节提供的小监控系统程序可以任意查看任意内存单元的内容，可以向任意 RAM 单元写入十六进制数据，使用方法与一般监控系统相同。

一、使用方法

小监控系统的使用方法如下：

1. 进入小监控系统

执行“CALL &6800”，就可由 F BASIC 工作状态进入小监控状态，画面上出现小监控提示符“*”，星号后的闪烁光标表示监控系统处于待命状态。

2. 查看内存

查看内存的命令格式为：

“＊■XXXX．YYYY”

——星号为小监控系统提示符；

——■为光标；

——XXXX 为查看内存的十六进制起始地址(1～4 位)；

——YYYY 为结束地址；

——“．”为查阅内存命令。

3. 写入内存

写入内存的命令格式为：

＊■XXXX:YY YY YY YY YY YY ……”

——XXXX 为写入内存的十六进制首地址；

——YY 为写入内存的十六进制数据，可一次写入八十个数据；

——“:”为写入内存命令。

4. 退出小监控系统

退出监控系统时只要键入“Q”后回车，就可返回 F BASIC 工作状态。

由于该监控系统只具备读、写内存的功能，故称之为小监控。

二、小监控系统程序清单

程序清单见 No. 4－9。

No. 4－9 6527 小监控系统

```
6800  2007AC  JSR  $AC07     ;令光标下移一行,位于行首
6803  A92A    LDA  #$2A      ;“＊”的 ASCII 码为 2AH
6805  2034AC  JSR  $AC34     ;送显监控系统提示符“＊”
```

```
6808  2043B3   JSR  $B343     ;扫描键盘,读入用户输入的内容,
                               直到按回车键后返回,用户输入
                               的内容从$501单元开始依次存
                               放。
680B  A000     LDY  #$00
680D  8490     STY  $90       ;输入内存检查开始,转$6837
680F  F026     BEQ  $6837
```

读、写内存的首、末地址置入地址指针单元

```
6811  A203     LDX  #$034     ;若是读内存则:
                               取首地址存入$8C、$8D
                               取末地址存入$8E、$8F
6813  0A       ASL
6814  0A       ASL
6815  0A       ASL            ;若是写内存则:
                               取首地址存入$8C、$8D
                               取写入数据存入$8E、$8F
6816  0A       ASL
6817  0A       ASL
6818  268E     ROL  $8E
681A  268F     ROL  $8F
681C  CA       DEX
681D  10F8     BPL  $6817
681F  A590     LDA  $90
6821  D004     BNE  $6827     ;标志单元为0时转置首址
                               不为0时表示已置首址
6823  B58F     LDA  $8F,X     ;转置首址
6825  958D     STA  $8D,X
6827  E8       INX
6828  F0F5     BEQ  $681F
```

```
682A  D011     BNE  $683D
      读到键盘缓冲区尾时的处理
682C  206B68   JSR  $686B      ;读、写内存判断处理
682F  4C0068   JMP  $6800      ;转重新输入
      从键盘缓冲区读到空格时,送显空格
6832  206B68   JSR  $686B      ;送显一个空格
      从键盘缓冲区读到命令符号后的判断处理
6835  A425     LDY  $25        ;取键盘缓冲区变址指针
      键盘缓冲区内容(用户输入的命令)的判断处理程序
6837  A200     LDX  #$00
6839  868E     STX  $8E
683B  868F     STX  $8F        ;预置末址初值
683D  B90105   LDA  $0501,Y    ;从键盘缓冲区读出一个数据
6840  C8       INY
6841  4930     EOR  #$30       ;取数与30异或
6843  C90A     CMP  #$0A
6845  90CA     BCC  $6811      ;是0～9转
6847  6988     ADC  #$88       ;不是0～9再加88+1(C)
6849  C9FA     CMP  #$FA
684B  B0C4     BCS  $6811      ;是A～F转
684D  8425     STY  $25        ;存键盘缓冲区变址指针
684F  C9B9     CMP  #$B9
6851  F0D9     BEQ  $682C      ;是缓冲区末转移
6853  C999     CMP  #$99
6855  F0DB     BEQ  $6832      ;是空格转移
6857  C9A7     CMP  #$A7
6859  F00B     BEQ  $6866      ;是"·"转移
685B  C993     CMP  #$93
685D  F007     BEQ  $6866      ;是":"转移
685F  C9EA     CMP  #$EA
```

```
6861  D09D     BNE   $6800      ;不是"Q"说明输入有错,转重新输
                                  入。
6863  4C1D82   JMP   $821D      ;是"Q"退出小监控,返回 F BASIC
6866  8590     STA   $90        ;置读、写存储器标志:
                                  读存储器时,$90=A7
                                  写存储器时,$90=93
6868  4C3568   JMP   $6835      ;继续读键盘缓冲区
```

读、写内存操作的判断与转口

```
686B  A590     LDA   $90        ;检查标志存储器
686D  F026     BEQ   $6895
686F  C9A7     CMP   #$A7
6871  F022     BEQ   $6895      ;是读内存转$6895
6873  C993     CMP   #$93
6875  D089     BNE   $6800      ;不是读、写内存,重新输入
```

数据写入内存处理

```
6877  A58E     LDA   $8E        ;取写入内存的数据于累加器
6879  A000     LDY   #$00       ;以 Y 变址
687B  918C     STA   ($8C,Y)    ;向内存写入一个数据
687D  4C8868   JMP   $6888      ;地址指针加 1 后返回
```

判断是否达到终址,地址指针加 1

```
6880  A58C     LDA   $8C
6882  C58E     CMP   $8E        ;地址低位比较,设置 C
6884  A58D     LDA   $8D
6886  E58F     SBC   $8F        ;地址高位比较,设置进位标志 C
6888  E68C     INC   $8C        ;地址指针加 1
688A  D002     BNE   $688E
688C  E68D     INC   $8D        ;地址低位有进位时,令地址高位加 1
688E  60       RTS
```

读内存、显示内存单元的内容,每行显示 8 个内存数据

```
688F  A58C     LDA   $8C
```

```
6891  2907     AND  # $07
6893  D00D     BNE  $68A2     ;一行不够8个数据不换行
6895  2007AC   JSR  $AC07    ;一行满8个数据后,令光标下移一
                              行,进行换行处理
6898  A58D     LDA  $8D
689A  20B468   JSR  $68B4     ;送显内存地址高位
689D  A58C     LDA  $8C
689F  20B468   JSR  $68B4     ;送显内存地址地位
68A2  A920     LDA  # $20
68A4  2034AC   JSR  $AC34     ;送显一个空格
68A7  A000     LDY  # $00
68A9  B18C     LDA  ($8C,Y)   ;从内存中读出一个数据
68AB  20B468   JSR  $68B4     ;送显一个内存数据
68AE  208068   JSR  $6880     ;读数地址指针加1,并与终址比较
68B1  90DC     BCC  $688F     ;未到终址则继续显示
68B3  60       RTS            ;已到终址则返回主程序
    送显数据子程序
68B4  48       PHA            ;送显数据压栈保存
68B5  4A       LSR            ;右移位
68B6  4A       LSR
68B7  4A       LSR
68B8  4A       LSR            ;把送显数据的高四位移至低四位
68B9  20BD7F   JSR  $68BD     ;送显十六进制数的高位
68BC  68       PLA            ;送显数据出栈,送显十六进制数的
                              低位
68BD  290F     AND  # $0F     ;屏蔽送显数据的高四位
68BF  0930     ORA  # $30     ;还原ASCII码
68C1  C93A     CMP  # $3A     ;累加器与立即数3A比较
68C3  9002     BCC  $68C7     ;数据为0～9时直接送显
68C5  6906     ADC  # $06     ;数据>9时,令ASCII码加7
```

```
68C7  4C34AC JMP   $AC34     ;送显累加器A中的字符
```

程序 No. 4－9 由于调用了系统的光标控制、送显、键盘扫描处理子程序，故程序短小精悍，功能较强。显而易见，程序 No. 4－9 只适用于 BS·2A 版本的 F BASIC 系统，如用于其它系统则要改变调用的几个系统子程序入口。

为便于使用，我们把 No. 4－9 安放在 $6800 开始的一段 RAM 区。

读者可把 No. 4－9 的数据编一个 F BASIC 写入程序，存入磁带或磁盘中，使用时，把它调入内存，运行一次后删除，即可使用小监控系统了。

注——键盘缓冲区中的数据经"EOR #$30"、再"ADC #$89"后，各数据与字符的对应关系如下：

0～9 对应的数据为 0～9；

A～F 对应的数据为 FA～FF。

"·"对应的数据为 A7；

":"对应的数据为 93；

空格对应的数据为 99。

第四节　汇编与 BASIC 程序结合实例

在 BASIC 语言的程序中，恰当地插入汇编语言程序，可以增强程序的功能，展现奇妙的强化效果。本节介绍的"音响动态演示八皇后"应用程序，就是一个汇编语言与 F BASIC 语言程序密切结合编程的实例。

一、八皇后问题

"八皇后"问题是著名数学家高斯提出来的。问题的内容是：在 8×8 的国际象棋盘中，放置八个皇后，使她们互相不

构成威胁(互不相吃,即在每一条横、竖、斜连线上,不得摆放两个皇后),如图 4—1 所示。

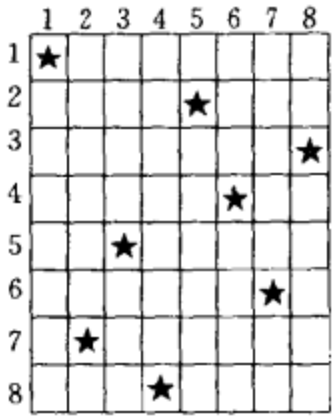


图 4—1　八皇后问题的解(之一)

图 4—1 示出了"八皇后"问题的其中一个解。要求编程演示其所有解的求解过程。

二、八皇后演示程序的功能、特点

关于"八皇后"问题的解题程序,中华学习机、PC 机上已有多次介绍。但这些程序只能给出 92 组答案,不能给出形象的解题过程。本求解、演示程序具有四个突出特点:

1. 使用不同配色的卡通图案"丽莎"饰演八个皇后,使皇后形象化;

2. 皇后的每一步移动,由动作卡通实现,使其从当前位置迅速跑到目标位置,从而使演示动感强烈。演示过程尤如 8 个演员在排练剧情;

3. 加入旋律不断变化的背景音乐伴奏和阶段效果音响,更进一步增强了演示过程的真实感。

4. 编程中巧妙地调用了 F BASIC 工作系统发声子程序,从而使程序短小精练,功能强大。

三、编程思路

1. 音乐演奏程序

在 BS·2A 版本的 F BAISC 工作系统中，有三段使用 $FC 单元控制演奏的音乐旋律，其中一段用于计算板，一段用于打字板，第三段没有使用；还有许多由 $FE、$FF 单元控制发声的效果音响(详见拙作《电脑游戏机硬件与编程特技》)，这些音乐旋律和效果音响我们都可调出使用。调用方法是：对发声控制开关 $FC、$FE、$FF 赋值，调用发声、演奏子程序即可(子程序的入口可选 $FD5B)。

2. 系统发声、演奏子程序的调用方法

凡进行过 BASIC 语言编程实践的朋友都知道，在 BASIC 环境下编写的程序是顺序执行的，当处理角色的控制时不能演奏音乐，而在演奏音乐时角色的动作要停下来。但任天堂游戏中，背景音乐的演奏、角色动作的控制、背景画面的移动则是同步进行的。这主要是它充分发挥了 6527 的中断服务功能而实现的。如把角色控制放在主程序中，而把音乐演奏程序放在中断服务程序中，这样就实现了同步控制的功能。

在八皇后程序中，欲加入背景音乐演奏，就必须把音乐演奏程序加入中断服务程序中。我们知道，BS·2A 版本 F BASIC 的中断服务程序入口地址为 $00ED，是一个 RAM 入口，系统复位后，该入口的程序为“JMP $880F”。因此，我们可以改变入口程序的转移地址，使中断服务首先转到演奏程序中，在音乐演奏处理完毕，再转到原中断服务程序。这样就加入了背景音乐的演奏功能。

3. 程序结构

音乐演奏程序用机器语言编写。程序分两段：一段用于改变中断程序入口；一段用于音乐演奏处理。解题、演示程序用 BASIC 语言编写，在 BASIC 程序中，根据发展需要设置不同的发声、演奏开关，以演奏不同的旋律和发出不同的效果音响。

四、机器语言程序清单

机器语言程序存放在 $6FE0 开始的 RAM 中，程序清单见 No.4－10、No.4－11。

音乐旋律演奏程序

No.4－10 中断服务程序

```
6FE0 48        PHA
6FE1 8A        TXA
6FE2 48        PHA
6FE3 98        TYA
6FE4 48        PHA              ;保护现场
6FE5 A9 FF     LDA    #$FF
6FE7 8D 15 40  STA    $4015     ;打开发声开关
6FEA 20 5B FD  JSR    FD5B      ;调演奏子程序
6FED 68        PLA
6FEE A8        TAY
6FEF 68        PLA
6FF0 AA        TAX
6FF1 68        PLA              ;恢复现场
6FF2 4C 0F 88  JMP    $880F     ;转系统中断服务程序
```

No.4－11 改变中断入口子程序

```
6FF5 A9 E0     LDA    #$E0
6FF7 85 EE     STA    $EE
```

```
6FF9 A9 6F      LDA      # $ 6F
6FFB 85 EF      STA      $EF
6FFD 60         RTS
```

上述机器语言程序的BASIC写入程序见No. 4－12。

No. 4－12 机器语言写入程序

```
POKE &6FE0,48,8A,48,98,48,A9,FF,8D,15,40,20,5B,
FD,68,A8,68,AA,68,4C,0F,88,A9,E0,85,EE,A9,6F,
85,EF,60
```

程序No. 4－12,既可以作为立即执行型程序,输入完毕回车,立即执行;也可以分给行号作为暂缓执行型程序。

程序No. 4－12运行后,再键入"CALL &6FF5"命令,即可把音乐演奏程序挂接在中断服务程序中。

五、F BASIC 程序清单

F BASIC 程序清单见No. 4－13。

```
10 DIM C(8),T(8),S(15),P(15): D=1:
   POKE 252,1 (由 $FC 控制演奏第一段旋律,慢节奏)
20 SPRITE ON: VIEW: GOSUB 500 (绘制棋盘)
30 POKE 252,3 (由 $FC 控制演奏第三段旋律,快节奏)
40 FOR I=0 TO 7: DEF MOVE(I)=SPRITE(1,3,1,1,0,I/3):
   POSITION I,20,30+I * 24: MOVE I
50 NEXT (定义八个皇后)
```

求出一组解后,音乐节奏变慢,同时由 $FF 控制发出求出一组解的效果音响。等待用户按任意键后,节奏变快,再演示下一组解。

```
60 K=1
70 IF T(K)=1 OR S(8+D－K)=1 OR P(D+K－1)=1 THEN 110
80 C(D)=K: T(K)=1: S(8+D－K)=1: P(D+K－1)=1: GOSUB
```

```
140
90 D=D+1: IF D>8 THEN TA=TA+1: LOCATE 24, 18:
   PRINTTA;: POKE 252,1,0,0.64: A$=INKEY$(0): POKE
   252,RND(2)+2: GOTO 120
100 GOTO 60
110 K=K+1: IF K<=8 THEN 70
```

未求出全部解时，继续进行；已求出全部解时打印总数，由 $FE 控制发出效果音响，延时一段时间后，关背景音乐，停机。

```
120 D=D-1: IF D<1 THEN SPRITE OFF: CLS: PRINT"TOTAL
   =";TA: POKE 254,32: PAUSE 500: POKE 252,0,0,0,0: END
130 K=C(D): T(K)=0: S(8+D-K)=0: P(D+K-1)=0: GOTO
   110
```

丽莎运动控制子程序

```
140 X=20+(K-1)*24: D1=D-1: Y=30+D1*24
150 X0=XPOS(D1): Y0=YPOS(D1)
160 IF X>X0 THEN DD=3: SS=(X-X0)/2: GOTO 180
170 DD=7 :SS=(X0-X)/2
180 DEF MOVE(D1)=SPRITE(1,DD,1,SS,0,2): POSITION D1,
   X0,Y0: MOVE D1
190 RETURN
```

绘制棋盘子程序

```
500 FOR I=1 TO 3: A$=A$+CHR$(255): B$=B$+CHR
   $(253): NEXT
510 FOR I=1 TO 8: FOR K=1 TO 3
520 FOR J=1 TO 4: PRINT A;B$: NEXT
530 IF I=8 AND K=3 THEN 550
540 PRINT: NEXT: SWAP A$,B$
550 NEXT
```

560 RETURN

六、使用说明

本程序只能在BS·2A版本的F BASIC系统中运行(所有的电脑学习机都配备该工作系统)。

运行程序后,在轻扬缓慢的背景音乐伴奏下,徐徐打印出背景画面,八位皇后成一列站在8×8的棋盘左边。随后背景音乐节奏加快,八位皇后分别在棋盘上匆忙奔跑,各自寻找自己的恰当位置。当一种摆放形式(一个解)排好后,八位皇后静止不动,发出排列结束的效果音响,同时背景音乐节奏变慢。这时,读者可核对答案的正确性,按任意键后,背景音乐节奏加快,八位皇后又急忙奔跑起来,进行下一组解的排列。

当92种排放形式全部演示完毕后,欢快的小号奏起"得胜"音乐。随后,一切音响消失,环境回归宁静,机器停止运行。

第五章　简单程序和分支程序设计

从本章开始，将分四章重点讨论汇编(机器)语言程序的设计方法和技巧。包括：简单程序设计、分支程序设计、循环程序设计、子程序设计、堆栈程序设计五个方面。

简单程序是复杂程序的基础，也是初学者程序设计入门的必修课；分支程序是设计循环程序的基础，各种各样的分支转移问题都将通过分支程序实现。因此，熟练掌握简单程序和分支程序的设计，是学习汇编语言的必经之路。

第一节　简单程序设计

简单程序也称顺序结构程序，它是自始至终按程序的顺序逐条执行的一类程序。这类程序在执行过程中没有分支、没有转移，平铺直叙，一竿到底，因而又称直线型程序。

简单程序的设计过程，就是按顺序编写程序。虽然程序设计方法比较简单，也同样含有数据输入、结果计算、结果输出、程序结束四个方面。也就是说，简单程序包括了程序设计中的每一部分，反映了程序设计的结构和全貌。

简单程序只能完成一些简单的操作，一般不单独使用。而且，任何一个程序也不可能单纯由简单程序组成，但初学者从简单程序设计入手确极为必要。本节将以示例的方式，介绍常用简单程序的形式和设计方法。实际上，本书的示例程序就是一个子程序库，其中的大多数程序可以作为工具程

序使用。

一、数据块转移

例 5－1 将地址 $6000～$6004 单元的数据转移到 $6100～$6104 单元中去。

由于转移数据较少，可直接写出程序，见 No. 5－1（程序从 $480 开始存放）。

No. 5－1 数据转移子程序

```
480 AD 00 60   LDA $6000
483 8D 00 61   STA $6100
486 AD 01 60   LDA $6001
489 8D 01 61   STA $6101
48C AD 02 60   LDA $6002
48F 8D 02 61   STA $6102
492 AD 03 60   LDA $6003
495 8D 03 61   STA $6103
498 AD 04 60   LDA $6004
49B 8D 04 61   STA $6104
49E 60         RTS
```

假定程序运行前，各单元的内容如下：

```
6000－ 01 02 03 04 05
6100－ FF FF FF FF FF
```

程序运行后，各单元的内容变为：

```
6000－ 01 02 03 04 05
6100－ 01 02 03 04 05
```

上述程序使用了取数、送数两条指令，反复进行同样的操作 5 次，把 $6000～$6004 单元的数据转移到 $610～$6104 单元。对于少量数据的转移，这种方法是可行的。当

需要转移的数据量较大时，程序 No. 5－1 显然是不经济的。

程序 No. 5－1 简单实用，即使在复杂的程序中，少量数据的转移场合也经常采用。

二、数据交换

例 5－2 将 $77 单元的数据 99H 与 $78 单元的数据 AAH 交换。

对处理数据交换问题的程序，设计方法很多，下面仅列举 3 种(程序从 $480 单元开始存放，下同)。

1. 借助一个内存单元作数据暂存器

若选择 $79 单元作为数据暂存器，编写的数据交换程序见 No. 5－2。

No. 5－2 数据交换程序(一)

```
480 A5 77 LDA  $77
482 85  79 STA  $79        ;($77)→($79)
484 A5 78 LDA  $78
486 85  77 STA  $77        ;($78)→($77)
488 A5 79 LDA  $79
48A85  78 STA  $78        ;($79)→($78)
48C 60 RTS
```

2. 使用堆栈暂存数据

使用堆栈作为数据暂存器的数据交换程序见 No. 5－3。

No. 5－3 数据交换程序(二)

```
480 A5 77 LDA  $77 ;($77)
482 48     PHA
483 A5 78 LDA  $78
485 85  77 STA  $77 ;($78)→($77)
487 68     PLA
488 85  78 STA  $78 ;($78)
```

48A 60 RTS

3. 使用不同的寄存器暂存数据

使用不同寄存器作为暂存器的数据交换程序见 No. 5－4。

No. 5－4 数据交换程序(三)

```
480 A6 77 LDX  $77 ;($77)
482 A4 78 LDY  $78 ;($78)
484 86  78 STX  $78 ;$78
486 84  77 STY  $77 ;$77
488 60     RTS
```

若上述三个程序运行前 $77＝99、$78＝AA。程序运行后 $77＝AA、$78＝99。

分析数据传送程序时，要紧紧抓住这样一个事实：数据从源地址传送到目标地址后，目标地址的内容被源地址的内容更新，而源地址的内容不变。这是分析一切传送指令的要领。

由上述三个程序可见，一道题目的解法是多样的，采用不同的方法、从不同的角度编程，可以强化思维、拓宽思路、熟悉指令，提高编程能力和技巧。

三、求反码

例 5－3 设 $65 单元的数据为 7AH，设计一个子程序求出它的反码存在 $66 单元中。

6527 指令系统中没有求反码的指令，但可以采用将 8 位二进制数与立即数 FFH 进行“异或”逻辑运算的方法，求出数据的反码。即：

7A＝01111010

异或　FF＝11111111
　　85＝10000101

求反码的程序见 No. 5－5。

No. 5－5 求反码子程序

```
480 A5 65   LDA $65
482 49 FF   EOR # $FF     ;7A EOR FF=85
484 85 66   STA $66       ;85→$66
486 60      RTS
```

运行程序 No. 5－5 后，$66 单元将存入 7AH 的反码 85H。

四、求补码

例 5－4 设 $60 单元的内容为 39H，编写一个子程序求出它的补码存入 $70 单元。

6527 指令系统中也没有求补码的指令，我们可通过对原码取反再加 1 就是补码的定义，编写出求补码的程序 No. 5－6。

No. 5－6 求补码子程序

```
480 A5 60   LDA $60
482 49 FF   EOR # $FF   ;A=39 EOR FF=C6
484 D8      CLD         ;D=0
485 18      CLC         ;C=0
486 69 01   ADC # $01   ;A+1=C7→A
488 85 70   STA $70     ;A=C7→$70
48A 60      RTS
```

运行程序 No. 5－6 后，$70 单元将存入 $60 单元内容的补码 C7H。

五、拆字

拆字又叫字的分解或分割。

例 5－5 设立即数为 57H，将其分成两段，每段 4 位，高 4 位置入 $80 单元、低 4 位置入 $81 单元，并使 $80、$81 单元的高 4 位清零。

即：($80)＝00000101＝05H

($81)＝00000111＝07H

在例 3－5 中，我们曾实践过把一个数据的低 4 位置入一个单元的低 4 位、其高 4 位清零，把数据的高 4 位置入另一个单元的高 4 位、其低 4 位清零。进行例 5－5 操作的程序与其大同小异，方法是：首先把屏蔽立即数高 4 位的数据置入 $81 单元，然后令原立即数连续右移 4 次，使其高 4 位移入低 4 位，而高 4 位自动清零，再把该数据置入 $80 单元。程序见 No. 5－7。

No. 5－7 拆字子程序

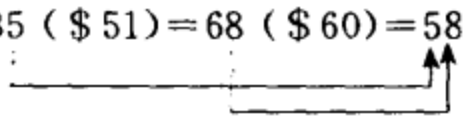
```
480 A9 57 LDA  # $57
482 29  0F AND  # $0F
484 85  81 STA  $81        ;57 AND 0F＝7→$81
486 A9 57 LDA  # $57       ;57 右移 4 次
488 4A     LSR
489 4A     LSR
48A 4A     LSR
48B 4A     LSR
48C 85  80 STA  $80        ;A＝05→$80
48E 60     RTS
```

运行程序 No. 5－7 后，$80＝05、$81＝07。

六、组字

组字又称字的组合。它表示从几个不同的单元或立即数中取出一定位数的二进制数，放到一个单元的不同位，以组成一个新字。

例 5－6 把 $50 单元的数据 35H 的低 4 位放入 $60 单元的高 4 位，把 $51 单元的数据 68H 的低 4 位放入 $60 单元的低 4 位。即：



设计本例题程序的要领是先拆字、后组字。程序见 No. 5－8。

No. 5－8 组字子程序

```
480 A5 50 LDA  $50      ;$50→A,A=35
482 0A    ASL
483 0A    ASL
484 0A    ASL
485 0A    ASL           ;A 左移 4 位,A=50
486 85 60 STA  $60      ;50→$60
488 A5 51 LDA  $51      ;($51)→A,A=68
48A 29 0F AND  #$0F     ;68 AND 0F=08
48C 05 60 ORA  $60      ;08 OR 50=58
48E 85 60 STA  $60      ;58→$60
490 60    RTS
```

七、求平均值

为简单起见，仅求两个数的平均值。

例 5－7 设两个单字节无符号立即数分别为 78H、FEH，求它们的平均值。

假定两立即数已放在 $60、$61 单元，平均值放在 $62 单元。程序见 No. 5－9。

No. 5－9 求两个单字节无符号数平均值子程序

```
480 A5 60   LDA  $60   ;($60)→A,A=78
482 18      CLC       ;C=0
483 65 61   ADC  $61   ;A+($61)=78+FE=76,C=1
485 6A      ROR       ;右转位，A=BB,C=0
486 85 62   STA  $62   ;A=BB→$62
488 60      RTS
```

程序 No. 5－9 中，因两数相加，可能有进位，故第 4 行程序采用循环右移，把进位 C 移入结果的最高位。

八、两字节地址指针加、减 1

在间接寻址方式中，数据的地址都是由两个存储单元表示的(地址的表示范围 0～FFFFH)，通常称为地址指针。而地址指针加 1、减 1 的操作，则是数据传送中使用最频繁的操作。

例 5－8 设取数地址指针为 $33、$34 单元，置数地址指针为 $35、$36 单元。编写地址指针加 1 操作子程序和减 1 操作子程序。

加 1 子程序见 No. 5－10，减 1 子程序减 No. 5－11。

No. 5－10 加 1 子程序(一)

```
480 A5 33   LDA  $33
482 18      CLC       ;C=1
483 69 01   ADC  #$01
485 85 33   STA  S33   ;$33+1+C→S33
```

```
487 A5 34    LDA  $34
489 69 00    ADC  #$00
48B 85 34    STA  $34        ;$34+0+C→$34
48D 60       RTS
```

N_O. 5－11 减 1 子程序(一)

```
490 A5 35    LDA  $35
492 38       SEC             ;C=0
493 E9 01    SBC  #$01
495 85 35    STA  $35        ;$35－1－C→$35
497 A5 36    LDA  $36
499 E9 00    SBC  #$00
49B 85 36    STA  $36        ;$36－$\overline{C}$→$36
49D 60       RTS
```

以上两程序的第 5、6、7 行是为了把低位地址加(减)1 的进位(借位)反映到高位地址中。

程序 N_O. 5－10 还可以写成 N_O. 5－12 的形式。

N_O. 5－12 加 1 子程序(二)

```
480 A9 01    LDA  #$01
482 18       CLC             ;C=1
483 65 33    ADC  $33
485 85 33    STA  $33        ;1+$33+C→$33
487 A9 00    LDA  #$00
489 65 00    ADC  $34
48B 85 34    STA  $34        ;0+$34+C →$34
48D 60       RTS
```

九、两字节加法

例 5－9 设两个 16 位二进制数据 3815H、42F1H，依次放在 $28～$2B 单元中(低位在前、高位在后，下同)。编写两

字节加法子程序，相加之和仍放在$28、$29单元。

程序见No.2－13。

No.2－13 两字节加法

```
480 A5 28 LDA  $28
482 18    CLC
483 65 2A ADC  $2A     ;低位相加
485 85 28 STA  $28     ;$28+$2A+C→$28
487 A5 29 LDA  $29
489 65 2B ADC  $2B     ;高位相加
48B 85 29 STA  $29     ;$29+$2B+C→$29
48D 60    RTS
```

程序No.2－13执行后，$28、$29单元的内容将为7B06H。

十、两字节减法

例5－10 设两个16位二进制数据7B06H、3815H、依次放在$28～$2B单元中。编写两字节减法子程序，相减之差仍放在$28、$29单元。

程序见No.2－14。

No.2－14 两字节减法

```
480 A5 28 LDA  $28
482 38    SEC
483 E5 2A SBC  $2A     ;低位相减
485 85 28 STA  $28     ;$28－$2A－C→$28
487 A5 29 LDA  $29
489 E5 2B SBC  $2B     ;高位相减
48B 85 29 STA  $29     ;$29－$2B－C→$29
48D 60    RTS
```

程序 No. 2－14 执行后，$28、$29 单元的内容将为 42FIH。

通过以上 10 例可见，简单程序所能处理的都是一些简单的问题，而对稍复杂的问题诸如：多个数据累加、大量数据传送、多个数据内容的交换、输入输出码的转换、字符串比较等问题，是不能胜任的。但简单程序没有分支、没有循环、使用的指令少、程序结构直观，既是初学者的入门阶梯，又是复杂程序的组成部分，还是要下功夫学好的。

第二节　分支程序设计

实际程序设计中，问题一般都比较复杂，像上节介绍的例题，仅是复杂问题中的一小部分，多数解决问题的程序常带有分支，以便根据不同的条件转移到不同的处理程序。

按给定的条件，对问题进行分析、比较、判断后决定程序的走向，是分支程序设计的主要内容。

6527 指令系统有众多的比较、转移指令，为分支程序设计提供了方便。我们可以根据问题的性质，选择不同的转移指令，控制程序的流向，以解决各种各样的分支问题。

应着重指出：高质量汇编程序的设计关键，就是灵活、准确地使用转移指令，特别是条件转移指令。高水平的程序设计技巧也往往从转移指令的使用中体现出来。

本节仍采取示例的方法，介绍各种分支程序的设计。

一、两字节地址指针加、减 1

例 5－8 曾给出用简单程序编写的两字节地址指针加、减 1 子程序 No. 5－10、No. 5－11、No. 5－12。下面再给出用

分支程序编写的两字节地址指针加 1 子程序 No. 5－15，减 1 子程序 No. 5－16。

No. 5－15 两字节地址指针加 1 子程序(三)

```
480 E6 33 INC  $33        ;低位加 1
482 D0 02 BNE  $486       ;$33≠0 转移
484 E6 34 INC  $34        ;$33=0 时令高位加 1
486 60    RTS
```

No. 5－16 两字节减 1 子程序(二)

```
490 A5 35 LDA  $35
492 D0 02 BNE  S496       ;$35≠0 转移
494 E6 36 DEC  $36        ;高位减 1
496 E6 35 DEC  $35        ;低位减 1
498 60    RTS
```

在程序 No. 5－15 中，首先进行低位指针加 1，若不等于 0，表示地址低位没有进位，直接返回；若地址低位指针等于 0，则表示低位有进位，故高位地址指针加 1。

在程序 No. 5－16 中，首先判断地址低位指针是否等于 0，若不等于 0 说明低位够减，则令地址低位指针减 1 后直接返回；若低位等于 0，说明地址低位减 1 将发生借位，故令地址高位减 1。

由以上两程序可见，分支程序的结构最紧凑、占用内存最少、运行效率最高。

二、单字节数据的比较

例 5－11 两个 8 位二进制数分别存放在 $28、$29 单元，$2A 作为标志单元。编一程序比较它们是否相等，若相等置标志 0；若不等置标志 FFH。

程序见 No. 5－17。

N_O. 5－17 单字节数据比较子程序(一)

```
480 A5 28   LDA $28
482 C5 29   CMP $29      ;$28－$29
484 D0 05   BNE $48B     ;结果不为0转移
486 A9 00   LDA #$00
488 85 2A   STA $2A      ;结果为0置00标志
48A 60      RTS
48B A9 FF   LDA #$FF
48D 85 2A   STA $2A      ;结果不为0置FF标志
48F 60      RTS
```

程序 N_O. 5－17 也可以写成 N_O. 5－18 的形式。

N_O. 5－18 单字节数据比较子程序(二)

```
480 A2 00   LDX #$00 ;0→X, X=0
482 A5 28   LDA $28
484 C5 29   CMP $29
486 F0 01   BEQ $489
488 CA      DEX          ;X－1→X, X=FF
489 86 2A   STX $2A
48B 60      RTS
```

程序 N_O. 5－18 中使用了变址寄存器 X 作为标志数据的存放单元,使程序节约了 4 个字节。

若采用 N_O. 5－19 的编程形式,还可以再节省 1 个字节。

N_O. 5－19 单字节数据比较子程序(三)

```
480 A5 28   LDA $28
482 45 29   EOR $29    ;$28 EOR $29
484 F0 02   BEQ $488   ;A等于0转移直接置标志0
486 A9 FF   LDA #$FF;A不等于0,置标志FF
488 85 2A   STA $2A
48A 60      RTS
```

程序 No. 5－19 中，两数据进行异或运算，若两数相等，累加器 A 中的内容为 0，所以可以转 $408 直接置相等标志 0，若两数不等，异或运算后，累加器 A 中的内容不等于 0，则置标志 FFH。

可见，使程序设计既实现要求的功能，又控制程序量最小是有许多途径可寻的。

三、判断奇、偶数

例 5－12 在 $B0 单元存放一个数据，编程判断其奇、偶性，若为奇数在 $B1 单元置入“01”标志；若为偶数则置入“02”；若为 0 则置标志“00”。

任何一个数（正数、负数、零），当转换为二进制数后，其 D0 位非 0 即 1，若原数是 0 或偶数时 D0＝0；是奇数时 D0 位一定等于 1。这个事实告诉我们，判断数的奇、偶性可通过判断 D0 位得到解决。因此，可以通过右移位、判断进位 C 的方法决定数的奇、偶性，程序见 No. 5－20。

No. 5－20 判断奇、偶数

```
480 A2 00   LDX  # $00   ;0→X,X=00
482 A5 B0   LDA  $B0
484 F0 05   BEQ  $48B    ;($B0)=0 转移,置 00 标志
486 E8      INX          ;X+1→X,X=01
487 4A      LSR          ;A 右移一位
488 B0 01   BCS  $48B    ;C=1 转移,置 01 标志
48A E8      INX          ;X+1→X,X=02
48B 86 B1   STX  $B1     ;标志置入标志存储器
48D 60      RTS
```

四、寻找两数中的最大数

例 5－13 两无符号数分别存放在 $6000、$6001 单元，

编程判断两数的大小，把较大的数放入 $6000 单元。

解决这一问题，首先进行数据大小的比较，若第一个数大或等于第二个数则直接返回；若第一个数小，还要进行两单元数据交换处理。判断无符号数的大小使用进位标志 C 较方便，C 标志可通过“CMP”指令设置（当执行“CMP”指令时，系统自动设定进位标志）。比较指令执行后，若 C＝1 表示累加器 A 中的数大或与比较的数相等；若 C＝0 则表示 A 中的数小。程序见 No. 5－21。

No. 5－21 寻找两数中的最大数

```
480 AD 00 60   LDA  $6000
483 CD 01 60   CMP  $6001
486 B0  0B     BCS  $494      ;若第一数大或相等则直接返回
489 48         PHA            ;若第二数大则第一数压栈
48A AD 01 60   LDA  $6001     ;取第二数
48D 8D  00 60  STA  $6000     ;大数存入 $6000
490 68         PLA            ;原第一数出栈
491 8D  01 60  STA  $6001     ;小数存入 $6001
494 60         RTS
```

五、判断正、负数

例 5－14 一个数（正数或负数）存放在 $6000 单元，编程判断该数的正、负，并给出正数标志 00、负数标志 01 在 $6001 单元。

由于本例只判断单字节数的正、负，故选用 BPL 指令较方便，当符号标志位 N＝1，表示该数为负数；若 N＝0 则表示为正数。N 状态可通过“LDA”指令的执行得到。程序见 No. 5－22。

No. 5－22 判断正、负数子程序

```
480 A2 00       LDX # $ 00   ;0→X
482 AD00 60     LDA $ 6000   ; $ 6000→A
485 10 01       BPL $ 488    ;N=0, X=0→ $ 6001
487 E8          INX          ;X+1→X, X=1
488 8E 01 60    STX $ 6001   ;X→ $ 6001
48B 60          RTS
```

六、两字节数据大小比较

例 5－15 设两字节无符号数分别存放在 $ 28、$ 29，$ 2A、$ 2B 单元中，比较它们的大小，置进位标志 C。若第一个数据小于第二个数据，置 C＝0；若第一个数据大于、等于第二个数据则置 C＝1。

两字节数据比较，应首先比较高位，若数 1 的高位小于数 2 的高位，则数 1 肯定小于数 2；若数 1 的高位大于数 2 的高位，则数 1 肯定大于数 2；当两数的高位相等时，低位大的数较大。根据这一思路编写的程序见 N_O. 5－23。

N_O. 5－23 两字节数据大小比较

```
480  LDA   $ 29
482  CMP   $ 2B     ;比较高位
484  BCC   $ 48E    ;数 1H<数 2H,C=0
486  BEQ   $ 48A    ;数 1H=数 2H,再比较低位
488  BCS   $ 48E    ;数 1H>数 2H,C=1
48A  LDA   $ 28
48C  CMP   $ 2A     ;比较低位
48E  RTS
```

程序 5－23 中使用了 3 条条件转移指令，实现了两字节数据大小比较、标志设置操作。工作原理如下：

首先，比较数据的高位，若数 1 的高位小于数 2 的高位，

则 C＝0，由第 3 条指令控制直接返回。在两数的比较中已自动设置了数 1 小于数 2 的标志 C＝0。

其次，若两数的高位相等(这时 C＝1)，则转第 6 行指令比较数据的低位。由比较低位的操作决定进位 C 的取值，若($28)≥($2A)则有 C＝1；若($28)＜($2A)则有 C＝0。也是自动给出标志。

第三、若 $29 单元的内容既不小于、又不等于 $2B 单元的内容，则必大于 $2B 单元的内容(C＝1)，必然数 1 大，故可直接返回。比较中已自动置 C＝1。

通过比较操作自动设置标志，是高明编程技巧的体现，希望大家能仔细体会其中的奥妙。

七、十六进制数据转换成 ASCII 码

例 5－16 把 $6000 单元的 8 位二进制数转换为可共显示的 ASCII 码，低位存入 $6001、高位存入 $6002 单元。

我们知道，电脑的字符输入与显示都是以 ASCII 码方式表示的，一位十六进制数据可以用代码 0～9、A～F 表示。它们对应的 ASCII 码分别为 30H～39H、41H～45H。

一个字节的十六进制数据转换成 ASCII 码要进行两项操作：首先进行拆字处理，把两位十六进制数拆成两个单位数；然后，再对 0～9 的数字加 30H、A～F 的数字加 37H。实际编程中，可对高、低位分别处理。实现程序见 No. 5－24。

No. 5－24 十六进制数据转换成 ASCII 码

```
480  AD 00 60   LDA   $6000    ;取十六进制数据到累加器 A
483  29  0F     ADN   #$0F     ;屏蔽高 4 位，只留低 4 位
485  C9  0A     CMP   #$0A
487  B0  06     BCS   $48F     ;大于 0AH 转移
```

```
189 18        CLC          ;C=0
48A 69 30     ADC  #$30    ;是 0～9，令 A+30H
48C 4C 92 04  JMP  $492    ;转移
48F 18        CLC          ;C=0
490 69 37     ADC  #$37    ;是 A～F，令 A+37H
492 8D 01 60  STA  $6001   ;低位数的 ASCII 码置入 $6001
495 AD 00 60  LDA  $6000
498 4A        LSR
499 4A        LSR
49A 4A        LSR
49B 4A        LSR          ;取高 4 位移入低 4 位
49C C9 0A     CMP  #$0A
49E B0 06     BCS  $4A6
4A0 18        CLC
4A1 69 30     ADC  #$30
4A3 4C AA04   JMP  $4A9
4A6 18        CLC
4A7 69 37     ADC  #$37
4A9 8D 02 60  STA  $6002   ;高位数的 ASCII 码置入 $6002
4AC 60        RTS
```

若在 $6000 单元置入数据 5FH，执行程序后，$6001 单元的内容将为 45H(F 的 ASCII 码)、$6002 单元的内容为 35H(5 的 ASCII 码)。

程序 No. 5－24 中，若把“ADC #$37”改为“ADC #$36”，可省略 4 条“CLC”指令。

你知道是什么理由吗？

八、十进制数转换成 7 段代码

例 5－17 编写显示十进制数字的 7 段显示器转换程序。

十进制数字放在$6000单元，转换后的7段显示器代码放在$6001单元。

7段显示器由发光二极管按图5－1所示的笔划排列组成。

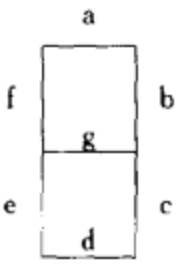


图5－1　7段显示器笔划排列示意图

通过控制7段笔划的亮、灭，可以显示0～9十个数字。如点亮“a、b、c、d、g”段可以显示“3”，点亮““a、c、d、f、g”段可以显示“5”等。

为方便控制各段笔划的亮、灭，令7段笔划分别与一位二进制数对应，这样可用7位二进制数表示7个笔划。对共阴极7段显示器来说，各段笔划约定的数位对应关系如图5－2所示。

十进制数 64 32 16 8 4 2 1

16进制数40 20 10 08 04 02 01

0	g	f	e	d	c	b	a
	6	5	4	3	2	1	0
各段的权	2	2	2	2	2	2	2

图5－2　7段笔划数位排列对应图

即：使用一个8位二进制数表示各段笔划，最高位恒为0，其余7位由低到高的对应顺序依次为“a、b、c、d、e、f、g”（“1”亮、“0”灭）。

例如：当显示十进制数字“3”时，应该“a、b、c、d、g”5段点亮（均为1），根据各段的权，有：

$2^0\times1+2^1\times1+2^2\times1+2^3\times1+2^6\times1$

$=1+2+4+8+64$

$=79D$

$=4FH$(十进制数“3”的代码)

“4FH”就是“3”的显示代码。

为方便使用，根据图 5－2 的排列，汇总 0～9 十进制数字对应的显示代码如表 5－1 所示。

表 5－1　共阴极 7 段显示器十进制数代码表

十进制数字	代　码	十进制数字	代　码
0	3F	5	6D
1	06	6	7D
2	5B	7	07
3	4F	8	7F
4	66	9	6F

编程思路是：从 $6000 单元取出数据，判断若为 0～9 的数字则把对应的显示代码送入 $6001 单元；若是其它数据则把“0”的显示代码送入 $6001。程序见 No. 5－25。

No. 5－25 十进制数据 7 段显示代码转换子程序(一)

```
480 A9 00      LDA  # $00     ;0→A
482 AE 00  60  LDX  $6000     ; $6000→X
485 E0 0A      CPX  # $0A
487 B0 03      BCS  $48C      ;X>9 转移，把 0 送入 $6001
489 BD 00  61  LDA  $6100,X   ;取对应的显示代码送 $6001
48c 8D 01  60  STA  $6001
48f 60         RTS
```

程序运行前，应先把 7 段显示代码置入 $6100：

6100－ 3F 06 5B 4F 66 6D 7D 07 7F 6F

运行程序，只要对$6000单元置入单位十进制数，就能向$6001送入显示代码。

若不考虑$6000单元十进制数据检查问题，程序No.5—25可以简化为No.5—26的形式。

No.5—26 十进制数据7段显示代码转换子程序(二)

```
480 AE 00 60 LDX  $6000
483 BD 00 61 LDA  $6100,X
486 8D 01 60 STA  $6001
489 60       RTS
```

通过以上几例可见，分支程序解决处理的问题比简单程序广泛的多，编程的技巧性、灵活性更比简单程序广泛。但是，学习、掌握分支程序的难度也较大，如何选择条件转移指令、选择什么样的条件转移、转移到哪里去等等，都不是一蹴而就的事情，必须经过反复的练习，才能运用自如。

学习分支程序设计，必须牢牢抓住标志寄存器各位状态这一关键环节。因为标志寄存器保存了指令执行后，结果是否为零、是正是负、是否有进位、是否有溢出等情况，这些情况都是分支程序设计中的重要判断依据。可以说，只要弄清了标志寄存器各位的来龙去脉，就打下了牢固的分支程序设计基础。

第六章　循环程序设计

在学习 BASIC 语言中，我们已经清楚了循环程序的格式、结构、功能、作用、循环嵌套等概念，也掌握了循环程序编写方法和技巧。

汇编语言中，同样有循环程序设计，同样有循环嵌套和多重循环。只不过因语言的特点不同，程序的结构、编程方法和技巧等略有不同罢了。但计算机编程方法是相通的，只要学会一种计算机语言，再学习其它语言都有触类旁通的功效。因此，只要注意区分汇编语言循环程序与 BASIC 循环程序的相同点、不同点，就可举一反三，很快驾驭汇编语言的循环程序设计。

第一节　循环程序的结构与特点

一、循环程序的结构

循环程序的结构包括以下四部分：

1. 循环初始化部分

循环初始化部分相当于 BASIC 中的“FOR TO STEP”语句，主要操作是建立计数器、地址指针和有关变量的初值。如累加器清零、置源数据块首址、置数据块长度、置内、外循环计数器、置循环次数等量的初值。

2. 循环处理部分

循环处理部分相当于BASIC语言的循环体。在这部分将进行实际的数据处理操作。它是循环的工作部分，由它控制完成重复执行程序段的内容。

3. 循环控制部分

循环控制部分相当于BASIC语言的“NEXT”语句，用在每一次循环处理后，以检查循环是否结束。若满足结束条件则转循环结束处理；若不满足结束条件，则修正计数器、调整地址指针，为下一轮循环做好准备。

4. 循环结束部分

该部分主要对循环处理结果进行分析、保存、显示或输出处理。

二、循环程序的特点

汇编程序的循环具有如下两大特点：

1. 特别适用于大块数据的传送

在处理动态数据块和多个数据块的传送中，一般采用变址寻址方式，通过改变变址寄存器内容的方法，达到改变存储器地址的目的。循环程序设计中，一般在每一轮处理结束后，将变址寄存器的内容加1，使它指向数据块的下一个元素，这样就可对下一个存储单元执行相同的操作，从而使循环程序段一次可以处理256个字节长度的数据块(因变址寄存器的字长为8位，最大取值为255)。对于长度大于256个字节的数据块，可以采用间接变址寻址方式，设置双重循环，内循环完成256个字节数据的传送，外循环控制基地址指针改变。这样，在每一次内循环(处理256个数据)操作结束后，令基地址指针加FFH(跳过已处理的256个数据)，使其指向下一256字节数据的首地址，就可以使用同一个内循环完成

任意长度的数据传送。

2. 循环程序必然包含分支

一般来说，分支程序可以不包含循环，但循环程序中必然包含分支。这是因为在循环程序的执行中，每一次循环操作结束，要做出继续循环或结束循环的判断，这就存在着控制循环的程序分支。若继续循环在调整循环指针后，由分支程序控制转移到循环入口；若结束循环也要由分支程序控制转移到新的程序段。因此，可以说："分支"是循环程序的控制中枢。6527 指令系统中的比较指令、条件转移指令为设计分支程序和循环程序奠定了基础。

第二节　数据块的传送

数据块的传送是计算机编程中最经常的操作，如定义卡通图案时，要向卡通定义页传送数个或上百个数据；绘制一幅全屏幕的背景画面，需要向 PPU 传送 960 个画面数据和 64 个画面配色数据，必要时还要传送 32 个 PPU 配色数据；演奏背景音乐时，更需要反复不停地向发声寄存器传送数据。因此，数据块传送操作，特别在游戏程序中占有很大的比重。本节将分单个数据的传送和多个数据块的传送两部分，介绍 6527 编程中常用数据块传送程序的编写方法。

一、单个数据块的传送

单个数据块传送中又有一页内数据块传送和多页数据块传送之分。当数据块长度限于一页(256 个字节)时，传送程序较简单，使用一个循环就可完成；当数据块长度大于一页时则要使用双重循环才能实现。

1. 一页内数据块顺序传送

例 6－1 将放在 $8000～$80FF 单元的数据传送到地址从 $6800 开始的存储单元中。

(1) 问题分析

本题虽然传送的数据量较多(256 个)，程序的设计思路却比较简单。即使用变址寄存器 X，取一个数据、送一个数据、X 加 1，指向下一个数据地址；再取、再送、X 再加 1……。直到 X 的取值从 0 再变到 0 时，256 个数据传送完成。

(2) 程序清单

No. 6－1 一页数据传送子程序(一)

```
6000 A2 00      LDX  # $00        ;计数器置初值
6002 BD 00 80   LDA  $8000,X      ;取数
6005 9D 00 68   STA  $6800,X      ;送数
6008 E8         INX               ;计数器加 1
6009 D0 F7      BNE  $6002        ;送完了吗？否，转移继续
600B 60         RTS               ; 是，结束返回
```

(3) 程序说明

程序 No. 6－1 中，第一行为送数计数器循环初始化部分，对变址寄存器 X 赋初值，基地址的偏址从 0 开始。

第二、三行为循环处理部分，它完成了取数、送数操作。

第四、五行为循环控制部分，首先令计数器 X 加 1；然后判断是否送数结束(X 是否等于 0)，若 X 不等于 0 则表示送数没有结束，由第五行分支转移到第二行继续送数；若 X 等于 0 表明送数结束，转入循环结束处理。

第六行为循环结束处理部分，由于本问题较简单，结束处理就是返回主程序。

2. 一页内数据逆序传送

例 6－2 将放在 $8000～$80FF 的数据按相反的顺序送到地址从 $6900 开始的单元中。

(1) 问题分析

由于数据按相反的顺序传送，源数据变址和目的数据变址必须使用两个寄存器才能实现。我们选 X 为源数据变址指针、Y 为目的数据变址指针，编写的程序见 No. 6－2。

(2) 程序清单

No. 6－2 一页数据传送子程序(二)

```
6000 A2 00      LDX  # $00      ;源数据变址器置初值
6002 A0 FF      LDY  # $FF      ;目的数据变址器置初值
6004 BD 00 80   LDA  $8000,X    ;取数
6007 9D 00 68   STA  $6800,Y    ;送数
600A 88         DEY             ;目的变址器加 1
600B E8         INX             ;源变址器加 1
600C D0 F7      BNE  $6002      ;送完了吗？否，转移继续
600E 60         RTS             ;是，结束返回
```

(3) 程序说明

程序 No. 6－2 中的循环初始化部分为第一、二行程序，因数据反序存放，故置数变址指针(计数器)Y 的初值为目的数据的最终数据地址变址值 FFH。

循环控制部分由第五、六、七行程序组成，目的数据控制变址器 Y 减 1，使目的数据从最后一个单元向前存放；源数据变址器 X 加 1，使源数据从第一个单元开始向后取出。并使用 X 作为送数计数器，当 X 等于 0 时，表示送数结束。

3. 不足一页数据的传送

需要特别说明的是：变址寄存器 X、Y 都是 8 位寄存器，在编程中常用来当做计数器使用，它们可以由指令控制而被置入初值，可以方便地用加 1(INX、INY)、减 1(DEX、

DEY)、比较(CPX、CPY)指令来修改、测试其内容，从而能够方便灵活地处理数据块、修改地址指针、控制循环次数等。

例 6－3 将 $8000～$807F 单元的数据顺序送到地址从 $6100 开始的单元中。

(1) 问题分析

传送数据不满一页，必须对计数器进行比较判断，确定结束循环处理的时机。程序见 No. 6－3。

(2) 程序清单

No. 5－3 不足一页数据传送子程序

```
6000 A2 00      LDX # $00      ;计数器置初值
6002 BD 00 80   LDA $8000,X    ;取数
6005 9D 00 61   STA $6100,X    ;送数
6008 E8         INX            ;计数器加 1
6009 E0 80      CPX # $80      ;送完了吗?
600B 90 F5      BCC $6002      ;否，转移继续
600D 60         RTS            ;是，结束返回
```

(3) 程序运行说明

当送数计数器 X 等于 7FH，把 $807F 单元的数据送至 $617F 单元后，执行第四行程序，X 加 1 等于 80H，执行“CPX”指令后进位位 C＝1，不满足分支转移的条件，故结束循环，返回主程序。

4. 大于一页数据块的传送

例 6－4 将 $8000～$87FF 单元中的数据传送到 PPU 地址 $1000 开始的单元中。

(1) 问题分析

这是一个背景字模数据传送问题，数据传送前必须先置 PPU 地址，方法是先向 $2006 单元送 PPU 地址的高字节，后

向$2006送PPU地址的低字节，然后依次向$2007单元置数就可以了。

由于传送的数据多于一页，应采取间接变址寻址方式。程序见No.6—4。

(2) 程序清单

No.6—4 向PPU传送多页数据子程序

```
6000 A9 10      LDA  #$10
6002 8D 06 20   STA  $2006
6005 A9 00      LDA  #$00
6007 8D 06 20   STA  $2006      ;置PPU地址为$1000
600A 85 F0      STA  $F0        ;置间接寻址地址低位指针
600C A9 80      LDA  #$80
600E 85 F1      STA  $F1        ;置间接寻址地址高位指针
6010 A0 00      LDY  #$00       ;一页计数器置0
6012 B1 F0      LDA  ($F0),Y    ;先间后变寻址取数
6014 8D 07 20   STA  $2007      ;送入PPU
6017 C8         INY             ;计数器加1
6018 D0 F8      BNE  $6012      ;不足一页继续
601A E6 F1      INC  $F1        ;送完一页后，令地址高位加1，
601C A5 F1      LDA  $F1         指向下一页首址
601E C9 88      CMP  #$88       ;检查是否送完所有数据
6020 90 EE      BCC  $6010      ;没送完继续
6022 60         RTS             ;全部送完，结束，子程序返回
```

(3) 程序说明

·程序No.6—4是一个双重循环程序，地址$6010～$6019程序为内循环程序，功能是间接变址寻址，取数送入$2007(PPU)。

·前七行程序是外循环的初始化部分，其中前四行程序

用于置 PPU 送数首地址；后三行程序用于设置间接寻址的地址指针，指针初值指向数据区首地址 $8000。

• 内循环是外循环的处理部分。

• $601A～$6021 程序为外循环的循环控制部分。当一页数据送完后(标志是：Y＝0)，令间接寻址地址指针高位加 1，使 $F0、$F1 指向下一页首址。如送数开始时，$F0、$F1 指向地址 $8000 单元，当送完一页数据，$F1＋1 后，则指向地址 $8100。当连续送完 8 页数据，在执行“INC $F1”后，$F1 单元的内容为 88H，不满足继续循环的条件，结束循环，返回主程序。

PPU 的地址设置和接受数据的操作方式很特殊，请充分注意这一点。

二、多个数据块的传送

例 6－5 编写子程序将 $9000～$900F 单元的第一个数据块送到 $6100～$610F 单元；将 $9100～$910F 单元的第二个数据块送到 $6200～$620F 单元；将 $9200～$920F 单元的第三个数据块送到 $6300～$630F 单元。

1. 问题分析

这是一个三个数据块的传送问题，由于是顺序搬移，而且数据块的长度相同(都是 16 个)，所以问题的求解过程不太复杂。但由于源数据块地址、目的数据块地址均不连续，故两个数据地址都必须采用间接寻址方式。数据块要一个个地送，送完一个数据块后只要改变取数、送数地址指针，就可以送另一个数据块，直到全部送完。因此，也必须使用双重循环结构编程。

2. 程序清单

选用$F0、$F1为取数地址指针、$F2、$F3为送数地址指针，编写的双重循环程序见No.6－5。

No.6－5 多块数据传送子程序

```
6000 A9 00   LDA   #$00
6002 85 F0    STA   $F0
6004 85 F2   STA   $F2
6006 A9 90   LDA   #$90
6008 85 F1   STA   $F1        ;置源数据地址指针$9000
600A A9 61   LDA   #$61
600C 85 F3   STA   $F3        ;置目的数据地址指针$6100
600E A0 00   LDY   #$00       ;变址计数器Y置0
6010 B1 F0   LDA   ($F0),Y   ;取数
6012 91 F2   STA   ($F2),Y   ;送数
6014 C8      INY              ;变址计数器Y加1
6015 C0 10   CPY   #$10       ;送够16个数据了吗?
6017 90 F7   BCC   $6010      ;不够转$6010继续
6019 E6 F1   INC   $F1        ;够了,源数据地址高位加1
601B E6 F3   INC   $F3        ;目的数据地址高位加1
601D A5 F3   LDA   $F3
601F C9 64   CMP   #$64       ;比较是否送完了三块数据
6021 90 EB   BCC    $600E     ;没有,转$600E,继续循环
6023 60      RTS              ;送完三块数据,结束
```

3. 程序说明

·前七行程序设置了取数地址指针$F0、$F1指向$9000；$F2、$F3指向$6100。

·$600E～$6018地址的程序为内循环，由该循环控制一次传送16个数据。

·$6019～$6022地址的程序是外循环的控制程序，当一组数据传送完毕，令$F1、$F3分别加1，使它们分别指向

下一块数据的源地址和目的地址首址。其中$601D～$6022地址的程序功能是：判断是否完成三块数据的传送。若($F3)=64则说明完成了三块数据的传送，于是退出循环，结束传送。

第三节　数据的查找

数据的查找也是计算机编程中经常碰到的问题，如在一段数据中查找最大的数据、查找最小的数据、查找具有某些特征的数据等等。特别是在排序、推理等类程序的编写中，数据的查找更是必不可少的组成部分。

例6－6 在一段数据中寻找最大的元素。设此段数据在$8000～$80EF单元，数据长度为240，查找的结果放在$F0单元，并设各数据均为无符号的8位二进制数。

1. 问题分析

该问题实际上是排序过程的一部分。可以这样安排程序：选取一个变址寄存器为变址指针，首先取第一个数据至累加器A，令A与第二个数据比较，若A大则令指针加1，与下一个数据比较；若第二个数据大则取大的数据放入A，令指针加1后再与下一个数据比较。如此一直比较到最后一个数据，累加器A中所存的数据即为最大数据。

2. 编写程序

根据以上思路编写的程序见No.6－6。

No.6－6 寻找最大数子程序

```
6000 A0 00     LDY   # $00      ;变址计数器Y置0
6002 A9 00     LDA   # $00      ;累加器A置初值
6004 D9 00 80  CMP   $8000,Y    ;A与存储器的数据比较
```

```
6007 B0 03      BCS   $600C      ;A 大转移,令 Y 加 1
6009 B9 00 80   LDA   $8000,Y    ;存储器中的数大则取到 A
600C C8         INY              ;Y 加 1
600D C0 F0      CPY   # $F0      ;所有数据比较完了吗?
600F 90 F3      BCC   $6004      ;没有,继续
6011 85 F0      STA   $F0        ;比较完了把最大数送入 $F0
6013 60         RTS              ;结束
```

3. 程序说明

程序 No. 6—6 虽然仅使用了单循环,程序较短小,但因两条比较指令和两条条件转移指令的混合判断处理,比以上介绍的双重循环程序还难阅读,特说明如下:

(1) 变址寄存器 Y 既是数据地址的变址指针,又是比较数据的计数器。循环开始,置 Y 等于 0。

(2)为便于比较程序的安排,先对累加器 A 置入初值 0,只要进行一次比较,累加器 A 即取入数据区的第一个数据。

(3) 在 A 与存储器的数据比较中,若 A 中的数大,则转移到地址 $600C,令变址寄存器 Y 加 1;若存储器中的数大,则取存储器中的数据于累加器 A,然后令 Y 加 1。

(4) 比较 Y 寄存器中的取值,若 Y<F0 则表明还未比较到最后一个数据,于是转 $6004 继续比较;若 Y=F0 则表明已比较完最后一个数据,故退出循环比较。

(5) 在整个循环比较过程中,累加器 A 中始终保存着当前所比较数中的最大数。

第四节　判断数据的类型

本节将以两个实例的编程介绍数据类型判断类程序的编

写方法。

一、确定负数的个数

例 6—7 已知带符号数据块放在 $A000～$A04F 单元中，编程找出其中的负数总数置入 $F0 单元。

1. 问题分析

(1) 选择条件转移指令的类型

确定数据块中的负数个数，就是找出 D7 位为 1 的所有数据。由于本例只有正数、负数两种类型，所以使用“BPL”指令或“BMI”指令较方便，因这两条指令以符号位 N 为执行依据，若 N＝0 则“BPL”条件满足，表示当前操作的数据为正数；若 N＝1 则“BMI”条件满足，表示为负数。因此，有两种方式编写分支程序。

(2) 选择取数指令的寻址方式

使用“LDA”指令，既可以完成取数操作，又可以影响标志寄存器的 N 标志位。同时，因数据块的长度小于 256 个字节，所以可以采用绝对 X 变址或绝对 Y 变址寻址方式取数。

(3) 循环计数器与负数计数器

循环计数器可以选择变址寄存器 X 或 Y，同时还可以使用它作为数据的变址指针。因问题中指定用 $F0 为负数存储器，我们可在程序运行中，把它作为负数计数器使用，这样可以节省一个计数器。

2. 编写程序

选择 X 作为变址指针、“BPL”条件转移指令编写的程序见 No. 6—7。

No. 6—7 确定数据个数子程序

```
6000 A2 00      LDX   # $00      ;变址寄存器置 0
```

```
6002 84 F0      STX   $F0        ;负数计数器置 0
6004 BD 00 A0 LDA   $A000,X  ;取数
6006 10 02      BPL   $600A      ;是正数转移
6008 E6 F0      INC   $F0        ;是负数令负数计数器加 1
600A E8         INX              ;变址指针加 1
600B E0 50      CPX   # $50      ;检查完所有数据了吗?
600D 90 F4      BCC   $6004      ;没有,转 $6004 继续
600F 60         RTS              ;检查完,返回
```

3. 程序说明

(1)开始两行程序对变址寄存器清 0,同时对负数计数器清 0。

(2) 当读出的数据为正数时,N=0,转 $600A 令 X 加 1;若读出的数据为负数,N=1,令负数计数器加 1,然后再令变址指针加 1。

(3) 根据计数器 X 的取值,判断是否检查完了所有数据,若没查完(X<50H)则转 $6004 继续检查;若已查完则结束循环,子程序返回。这时,$F0 中存储的就是负数的个数。

4. 讨论

若要确定数据块中的正数个数,只要稍微改动程序 No. 6－7 的个别指令即可。你能做到吗?

二、确定正数、负数和零数的个数

例 6－8 设带符号数据块放在 $B000～ $B098 单元中,编程找出其中的负数总数置入 $F0 单元、正数的总数置入 $F1 单元、零数的总数置入 $F2 单元。

1. 问题分析

本问题仅是在例 6－7 中,增加了正数和零数的检查,因此只要在程序 No. 6－7 的基础上,增加检查正数、零数和累

计正数、零数个数的处理，即可得到例 6—8 的处理程序。

实际上，对取到累加器 A 中的数据，只要筛选出零数和负数，剩下的就是正数了。因此，在累加器从存储器中读出数据后，可先使用“BNE”或“BEQ”指令控制零数的累加；然后再判断是否为负数，若是负数则令符号计数器加 1；否则令正数计数器加 1。

2. 编写程序

根据以上分析，选择 X 作为变址指针、“BPL”、“BNE”条件转移指令编写的程序见 No. 6—8。

No. 6—8 确定正数、负数、零数子程序

```
6000 A2 00      LDX  # $00      ;变址寄存器置 0
6002 84 F0      STX  $F0        ;负数计数器置 0
6004 84 F1      STX  $F1        ;正数计数器置 0
6006 84 F2      STX  $F2        ;零数计数器置 0
6008 BD 00 A0 LDA  $B000,X  ;取数
600B D0 05      BNE  $6012      ;数据不为零转移
600D E6 F2      INC  $F2        ;数据为零，令零数计数器加 1
600F 4C 1B 60 JMP  $601B      ;转移，令 X 加 1
6012 10 05      BPL  $6019      ;是正数转移
6014 E6 F0      INC  $F0        ;是负数令负数计数器加 1
6016 4C 1B 60 JMP  $601B      ;转移，令 X 加 1
6019 E6 F1      INC  $F1        ;是正数令正数计数器加 1
601B E8         INX             ;变址指针加 1
601C E0 99      CPX  # $99      ;检查完所有数据了吗？
601E 90 EA      BCC  $6008      ;没有，转 $6008 继续
6020 60         RTS             ;检查完，返回
```

3. 程序说明

程序 No. 6—8 比 No. 6—7 的结构复杂的多，现简单说明

如下：

(1) 当累加器 A 中的数据等于 0 时，执行 $600D 地址的指令，令零数计数器 $F2 加 1，然后转 $601B，令 X 加 1。

(2) 若 A 中的数据非 0 时，转 $6012 检查是否为正数，若是负数则执行 $6014 地址的指令，令负数计数器 $F0 加 1，再转 $601B，令 X 加 1。

(3) 若 A 中的数据为正数时，则执行 $6019 地址的指令，令正数计数器 $F1 加 1，再执行 $601B 的指令，令 X 加 1。

(4)当变址指针 X＝99H 时，所有的数据检查完毕，结束检查，返回主程序。

第五节　数据的累加

例 6－9 一串无符号的二进制数据放在 $C000～$C03F 单元，编程求出它们的累加和。

1. 问题分析

因是多个数据相加，在每一次运算中，必须考虑进位问题。

因数据量比较小(仅有 40H 个数据)，累加和使用两个字节即可完全存放。

程序结构为：采用单循环，用变址寄存器 X 作为计数器，选 $F0 单元存和的低位、$F1 存高位。进行一次求和运算后，检查进位位 C，若无进位继续累加下一个数据；若有进位则令 $F1 加 1。

2. 编写程序

根据以上分析，编写程序见 No. 6－9。

No. 6－9 数据累加子程序(一)

```
6000 A2 00      LDX   # $00      ;计数器清零
6002 86 F0      STX   $F0
6004 86 F1      STX   $F1        ;存和单元清零
6006 A5 F0      LDA   $F0        ;取和的低位
6008 18         CLC              ;C=0
6009 7D 00 C0   ADC   $C000,X    ;加一个数据
600C 90 02      BCC   $6010      ;无进位转移
600E E6 F1      INC   $F1        ;有进位令和的高位加 1
6010 85 F0      STA   $F0        ;存和的低位
6012 E8         INX              ;计数器加 1 指向下一个数据
6013 E0 40      CPX   # $40      ;累加完了吗?
6015 90 EF      BCC   $6006      ;没有,继续
6017 60         RTS              ;结束
```

程序 No. 6－9 结构比较简单，不多分析。为开拓思路，下面给出例 6－9 问题的另一形式的程序，见 No. 6－10。

No. 6－10 数据累加子程序(二)

```
6000 A2 3F      LDX   # $3F      ;计数器置入数据总个数
6002 A9 00      LDA   # $00
6004 85 F1      STA   $F1        ;存和的高位单元清零
6006 18         CLC              ;C=0
6007 7D 00 C0   ADC   $C000,X    ;从最后一个数据向前累加
600A 90 02      BCC   $600E      ;没有进位转移
600C E6 F1      INC   $F1        ;有进位令和的高位加 1
600E CA         DEX              ;计数器减 1
600F 10 F5      BPL   $6006      ;数据未累加完继续
6011 85 F0      STA   $F0        ;数据累加完存和的低位
6013 60         RTS
```

程序 No. 6－10 采取了从最后一个数据向前累加的方法，由计数器 X 减 1 至 X=FFH 时，退出循环。

累加和的低位始终存在累加器A中，仅在有进位时才把进位累加至$F1单元，直到累加结束后才存和的低位。

由于程序省略了对$F0单元清零和X与立即数比较的操作，使程序量压缩了4个字节。

程序No.6－10中的循环体仅占11个字节，比No.6－9少用了6个字节，故程序No.6－10的执行速度将比No.6－9快三分之一。数据量越大，其优越性越明显。

说明：程序No.6－10只适用于累加数据在80个以内的情况。当数据量超过80个时，应怎样修改程序；当数据量超过一页时应怎样编写程序，你能找出诀窍吗？

限于篇幅，有关循环程序的设计仅介绍以上几例。现把循环程序设计的有关内容总结如下：

(1) 循环程序的结构

循环程序的结构包括：循环初始化、循环处理、循环控制和循环结束四个组成部分。各部分有机结合、协调工作、缺一不可。

(2) 循环与分支

循环包含分支、分支控制循环，两者互相联系、互相制约。灵活而准确地使用比较指令，特别是条件转移指令，是设计高质量循环程序的基本功。

(3) 循环程序的嵌套

循环程序的嵌套是指在一个循环处理程序内，包含另一个循环程序、甚至数个循环程序。但应注意内、外循环的层次要清楚，允许几个循环并列使用，但不允许交叉使用。

(4) 循环程序的任务

循环程序的任务在于完成大量的、处理形式完全相同的、重复性的计算工作。它几乎能够解决形式各异的各类课题，是机器语言程序设计的核心和精华。

第七章　子程序设计

在计算机编程中，通常把那些在处理功能上具有一定的独立性、能够完成某些特定的运算和操作、并被经常调用的程序段抽象成一个独立的程序，如：代码转换、数据传送、字符显示、函数运算等，这个独立的程序就被称为“子程序”。

子程序是结构化程序设计的核心，是优化程序结构的有效工具。使用它可以简化程序设计、提高编程效率、清晰程序结构；可以方便程序调试、修改和维护；可使程序易于阅读、便于交流。因此，子程序也是计算机程序设计中的必修课之一，必须下功夫掌握它。

第一节　子程序的概念

一、转子和返主

子程序为系统所公有，可被系统中不同的程序段所调用。一般把调用子程序的程序称为主程序，把调用子程序的操作称为“转子”，把子程序执行结束返回调用程序的操作称为“返主”。

多数程序都是由一个主程序和数个子程序组成。在程序的适当位置安排子程序的调用点，通过转子指令“JSR”和返主指令“RTS”，可以实现主程序对子程序的调用和子程序到主程序的自动返回。

二、转子和返主操作

为了保证子程序准确地返回主程序，要求在转子前必须把断点状态保存起来，即保护现场。转子指令“JSR”具有自动保护现场的功能，当执行“JSR”指令时，系统将自动从程序计数器 PC 中读出断点地址(即转子指令的下一条指令地址)减一后送入堆栈保存，而把“JSR”指令指定的子程序地址送入程序计数器，使 CPU 转向执行子程序。

当子程序执行结束，执行到返主指令“RTS”时，系统将自动从堆栈中取出断点地址加一后置入程序计数器 PC，使 CPU 转向到“JSR”指令后的下一条指令处执行程序。

由此可见，“JSR”和“RTS”指令的作用就是转移程序的控制权。“JSR”把程序的控制权转交给子程序；“RTS”则把程序的控制权再交还给主程序。

三、子程序的嵌套

在子程序中又可安排子程序的调用过程，称为子程序的嵌套。6527 允许子程序多层嵌套，即子程序一层套一层。子程序嵌套的层数称为子程序嵌套深度，嵌套深度由程序运行中，对系统堆栈的占用状况决定，一般允许嵌套十几层。

子程序嵌套中要注意处理好各层次之间的返回。实际编程中既可以逐层返回，也可以跳跃返回，这涉及到高层次的编程技巧问题。初学者应仍以逐层返回为主攻点。本章仍采用示例的方式，介绍基本子程序的设计方法。

四、子程序设计的基本要求

为提高子程序的公用性，一般要求把子程序设计成通用

方式，以便为不同的程序服务。因此，对子程序设计提出两点基本要求：第一避免绝对寻址方式，以适用于存储在所有内存地址范围的数据；第二尽量采用相对跳转而避免绝对跳转，以提高子程序的灵活性，使子程序具有工具性质，可被任何程序直接引用。

第二节　单个子程序的设计

本节仅介绍单个子程序的设计方法。

一、数据块的传送

例 7—1 采用子程序结构，编写例 6—4 的数据传送子程序。

在循环程序设计中，例题 6—4 的处理程序为 No. 6—4。其中的内循环程序结构独立、功能专一，可把它从主程序中移出，抽象成通用的一页数据传送子程序，见程序 No. 7—1。

No. 7—1 向 PPU 传送多页数据子程序

```
6000 A9 10     LDA   # $10
6002 8D 06 20  STA   $2006
6005 A9 00     LDA   # $00
6007 8D 06 20  STA   $2006     ;置 PPU 地址为 $1000
600A 85 F0     STA   $F0       ;置间接寻址地址低位指针
600C A9 80     LDA   # $80
600E 85 F1     STA   $F1       ;置间接寻址地址高位指针
6010 20 1C 60  JSR   $601C     ;向 PPU 送入一页数据
6013 E6 F1     INC   $F1       ;送完一页后，令地址高位加 1，
6015 A5 F1     LDA   $F1       指向下一页首址
6017 C9 88     CMP   # $88     ;检查是否送完所有数据
```

```
6019 90 EE      BCC   $6010      ;没送完继续
601B 60         RTS              ;全部送完,结束,子程序返回
```

No. 7－1－1 向 PPU 送入一页数据子程序

```
601C A0 00      LDY   #$00       ;一页计数器置 0
601E B1 F0      LDA   ($F0),Y    ;先间后变寻址取数
6020 8D 07 20   STA   $2007      ;送入 PPU
6023 C8         INY              ;计数器加 1
6024 D0 F8      BNE   $601E      ;不足一页继续
6026 60         RTS
```

程序 No. 7－1 中，$601C～$6026 为向 PPU 送入一页数据子程序，子程序的入口地址为 $601C。当设置取数地址指针后，调 $601C 就可完成向 PPU 送入 256 个数据的操作。子程序返回后，执行 $6013 以后的程序，调整取数地址指针，判断是否完成数据传送任务，若没完成，则转 $6010，继续调用 $601C 子程序。因此程序 No. 7－ 1 具有 No. 6－4 相同的功能。

另外，比较程序 No. 6－4 可以看出：程序 No. 7－1 比 No. 6－4 有更大的灵活性，而且其中的向 PPU 送数子程序 No. 7－1－1，不仅可供 No. 7－1 调用，也可以供任意主程序调用。只要设置 PPU 地址、对 $F0、$F1 单元置入数据区首址后，就可调用它向 PPU 送入一页数据。如果调用前对 Y 寄存器置入初值，从子程序的第二条指令入口（即调用 $601E），则可向 PPU 送入 256 以内的任意个数据。

二、判断数据类型的程序设计

例 7－2 数据块 1 放在 $8000～$80FF 单元、数据块 2 放在 $8100～$8148 单元、数据块 3 放在 $$8180～$81FF 单元，编程统计各数据块的正数、负数、零数个数，分

别存入 $F7～$FF 单元。

1. 通用数据类型统计子程序的设计

关于寻找正数的实现程序，已在循环程序设计中进行了详细讨论。现在仅讨论子程序的设计。

由于三个数据块所进行的是同一项操作(统计数据类型的个数)，对照程序 No. 6－8，可将数据类型统计程序段 $6008～$601E 抽象成子程序。为使子程序具有通用性，应把绝对 X 变址寻址方式改为 Y 间接变址寻址方式。另外，因三个数据块的数据类型计数器均为连续的零页单元，所以可以使用零页 X 变址寻址方式，以减小程序量。

根据以上的思路，编写的数据类型统计通用子程序见 No. 7－2。

No. 7－2 数据类型统计通用子程序

```
6100 B1 F0     LDA ($F0),Y    ;取数
6102 D0 05     BNE $6109      ;数据不为零转移
6104 F6 F9     INC $F9,X      ;数据为零,令零数计数器加 1
6106 4C 1B 60  JMP $6112      ;转移,令 Y 减 1
6109 10 05     BPL $6110      ;是正数转移
610B F6 F8     INC $F8,X      ;是负数令负数计数器加 1
610D 4C 1B 60  JMP $6112      ;转移,令 Y 减 1
6110 F6 F7     INC $F7,X      ;是正数令正数计数器加 1
6112 E8        DEY            ;变址指针加 1
6113 E0 99     BNE $6100      ;没检查完,继续
6115 60        RTS            ;检查完,返回
```

2. 浮动地址的数据类型统计通用子程序设计

读者在看懂了程序 No. 7－2 的程序结构后，可把它改为浮动地址的通用子程序 No. 7－3。因避免了绝对跳转，该子程序可以放在任意地址单元中使用。

N_O. 7－3 浮动地址的数据类型统计通用子程序

```
6100 B1 F0   LDA   ($F0),Y      ;取数
6102 D0 04   BNE   $6108        ;数据不为零转移
6104 F6 F9   INC   $F9,X        ;数据为零,令零数计数器加 1
6106 D0 08   BNE   $6110        ;转移,令 Y 减 1
6108 10 06   BPL   $610E        ;是正数转移
610A F6 F8   INC   $F8,X        ;是负数令负数计数器加 1
610C D0 02   BNE   S6110        ;转移,令 Y 减 1
610E F6 F7   INC   $F7,X        ;是正数令正数计数器加 1
6110 E8      DEY                ;变址指针加 1
6111 E0 99   BNE   $6100        ;没检查完,继续
6113 60      RTS                ;检查完,返回
```

该子程序的调用要求如下：

(1) 调用前把数据区首址置入 $F0、$F1 单元。

(2) 从 $F7 开始，每连续的三个单元为正、负、零数个数的存储单元，调用前必须把它们清零。

(3) 变址寄存器 X 决定数据类型存储单元的偏址，取值为 0、3、6。

(4) Y 寄存器既是变址寻址控制器，又是计数器，调用前必须根据数据区长度设置恰当的初值。

3. 三个数据块数据类型统计程序设计

有了数据类型统计通用子程序，就可以安排主程序了。

把数据块的首地址置入 $F0、$F1，向数据块长度计数器 Y 置入初值，调用子程序就可以统计出各类数据的个数。不过在调用子程序之前，必须把数据块的序号乘 3(0、3、6)置入 X 寄存器，以便把各数据块的数据类型个数置入各自的计数器。

另外，因子程序 N_O. 7－3 的循环退出判断由 Y＝0 决定，

因此，必须合理设置 Y 的初值，才能保证不丢失数据块中应检查的数据。

根据以上思路编写的三个数据块数据类型统计程序见 No. 7－4。

No. 7－4 三个数据块数据类型统计程序

```
6000 A2 09      LDX   # $09
6002 A9 00      LDA   # $00
6004 95 F7      STA   $F7,X
6006 CA         DEX
6007 10 FB      BPL   $6004     ;置$F7～$FF=0
6009 AA         TAX             ;X=0
600A 85 F0      STA   $F0
600C A9 80      LDA   # $80
600E 85 F1      STA   $F1       ;置数据1的首地址于$F0、$F1
6010 A0 00      LDY   # $00     ;置计数器Y初值
6012 20 00 61   JSR   $6100     ; 统计数据块1的数据类型置入
                                  $F7、$F8、$F9
6015 A2 03      LDX   # $03     ;X=3
6017 A9 FF      LDA   # $FF
6019 85 F0      STA   $F0
601B A9 FF      LDA   # $80
601D 85 F0      STA   $F1       ;$F0、$F1指向$80FF
601F A0 49      LDY   # $49     ;数据块2的数据长度置入Y
6021 20 00 61   JSR   $6100     ; 统计数据块2的数据类型置入
                                  $FA、$FB、$FC
6024 A2 06      LDX   # $06     ;X=6
6026 A9 7F      LDA   # $7F
6028 85 F0      STA   $F0
602A A9 81      LDA   # $81
```

```
602A 85 F1     STA   $F1        ;$F0、$F1 指向 $817F
602C A0 80     LDY   #$80       ;数据块 3 的数据长度置入 Y
602E 20 00 61  JSR   $6100      ; 统计数据块 3 的数据类型置入
                                  $FD、$FE、$FF
6031 60        RTS
```

4. 三个数据块数据类型统计程序说明

程序 No. 7—4 较难懂，现说明如下：

（1）$6000～$6008 为计数器清零程序段，由它完成对 $F7～$FF 单元的清零。

（2）$6009～$6014 为数据块 1 数据类型统计处理程序。因该块数据块首址为 $8000，数据长度为 256 个字节，$F0、$F1 指向 $8000。置计数器 Y＝0 的理由是：当 Y＝0 调子程序 No. 7—3，首先检查 $8000 单元的数据；然后在执行 $6110 地址的指令时，令 Y 减 1，Y＝FF，其后可由循环程序控制，从 $80FF 开始依次向前检查数据；直到 Y＝1，检查完 $8001 单元后，Y 减 1 等于 0，结束循环，返回主程序。因此，完成了数据块所有数据的检查。如果令 Y＝FF 调用子程序则将遗漏 $8000 单元的检查（你知道是什么原因吗）。

（3）在调用子程序中，因 X＝0，故把数据块 1 的正数、负数、零数的个数分别置入了 $F7、$F8、$F9 单元。

（4）$6015～$6023 程序段完成数据块 2 的类型统计。因数据块 2 的首址为 $8100，数据长度为 49H，由子程序的特点出发，令地址指针 $F0、$F1 指向 $80FF，令 Y＝49。这样做的理由是：在调用子程序中，首先检查 $80FF＋49＝$8148 单元的内容，然后依次向前检查，直到 Y＝1，检查完 $8100 单元的内容后，因 Y 减 1 等于 0，结束循环，返回子程序。但如果令 Y 等于 48、地址指针指向 $8100，将遗漏

$8100 单元的数据。因现在 X＝3，故数据块 2 的正数、负数、零数个数将分别存入 $FA、$FB、$FC 单元。

(5) $6024～$6030 程序段的作用是处理数据块 3。因该块数据区的首址为 $8180、数据长度为 80H 个，故令地址指针指向 $817F、令 Y＝80 调子程序。你能分析出是什么原因吗？

现在因 X＝6，故数据块 3 的正数、负数、零数个数分别置入 $FD、$FE、$FF 单元。

三、寻找最大值程序的设计

例 7－3 无符号数据块在 $A000～$A088，编写程序找出其中的最大数存入 $FF 单元。

寻找最大值的问题我们已在循环程序设计中进行了详细讨论，程序 No. 6－6 虽然简单易懂，但通用性不好，如数据的存储区发生改变，就要对程序进行修改。下面将给出求最大值的通用程序。

1. 求最大值的通用子程序设计

因子程序具备通用性的要求之一就是避免绝对寻址，因此必须把程序 No. 6－6 中的绝对寻址方式改为间接寻址方式，仍取 $F0、$F1 为间接地址指针，编写的子程序见 No. 7－5。

No. 7－5 寻找最大数子程序

```
6100 A9 00    LDA  # $00      ;累加器 A 置初值，A＝0
6102 A8       TAY             ;Y＝0
6103 D1 F0    CMP  ($F0),Y    ;A 与存储器的数据比较
6105 B0 02    BCS  $6109      ;A 大转移，令 Y 加 1
6107 B1 F0    LDA  ($F0),Y    ;存储器中的数大则取到 A
```

```
6109 C8        INY              ;Y 加 1
610A C4 F2     CPY   $F2        ;查完所有数据了吗?
610C D0 F5     BNE   $6103      ;没查完,继续
610E 60        RTS              ;查完了,结束
```

该子程序的调用条件:

(1) 调用前将数据区首址置入 $F0、$F1 单元,数据区长度置入 $F2 单元。

(2) 子程序所用的内存单元有 $F0、$F1、$F2,所用的寄存器有 A、Y。

2. 求例 7—3 最大值子程序的设计

根据子程序 N_0. 7—5 的要求,编写求最大值的子程序见 N_0. 7—6。

N_0. 7—6 寻找 $A000～$A088 中最大的数据

```
6000 A2 88     LDX  #$88
6002 86 F0     STX  $F0      ;置数据区地址低位
6004 A2 89     LDX  #$89
6006 86 F2     STX  $F2      ;置数据区长度
6008 A9 A0     LDA  #$A0
600A 85 F1     STA  $F1      ;置数据区高位
600C 20 00 61  JSR  $6100    ;寻找最大数存于 A
600F 85 FF     STA  $FF      ;最大数送入 $FF
6011 60        RTS
```

第三节　多个子程序的设计

一个主程序内可以包含多个并列的子程序,该并列的子程序中还允许嵌套另外的子程序。因此,多个子程序的程序结构也是树状结构。

一、三个数据块数据类型统计程序的优化

程序 No. 7—4 通过调用子程序 $6100，完成了三个数据块数据类型统计的任务。其实 No. 7—4 的前五行程序(对各计数器清零)也可以独立成计数器清零子程序，供其它数据类型统计程序调用，这样可使主程序的结构更清晰。独立出的子程序见 No. 7—7。

No. 7—7 $F7～$FF 清零子程序

```
6120 A2 09   LDX   #$09
6122 A9 00   LDA   #$00
6124 95 F7   STA   $F7,X
6126 CA      DEX
6127 10 FB   BPL   $6004    ;置$F7～$FF=0
6129 60      RTS
```

该子程序运行中仅使用 A、X 寄存器。

在弄清程序 No. 7—4 的编程原理后，可将它优化成 No. 7—8 的形式。

No. 7—8 三个数据块数据类型统计程序

```
6000 20 20 61  JSR   $6120   ;置$F7～$FF=0
6003 AA        TAX          ;X=0
6004 85 F0     STA   $F0
6006 A9 80     LDA   #$80
6008 85 F1     STA   $F1     ;置数据1的首地址于$F0、$F1
600A A0 00     LDY   #$00    ;置计数器Y初值
600C 20 00 61  JSR   $6100   ;统计数据块1的数据类型置入$F7、
                               $F8、$F9
600F A2 03     LDX   #$03    ;X=3
6011 A9 FF     LDA   #$FF
```

```
6013 85 F0      STA   $F0      ;$F0、$F1 指向 $80FF
6015 A0 49      LDY   #$49     ;数据块 2 的数据长度置入 Y
6017 20 00 61   JSR   $6100    ;统计数据块 2 的数据类型置入 $FA、
                               $FB、$FC
601A A2 06      LDX   #$06     ;X=6
601C A9 7F      LDA   #$7F
601E 85 F0      STA   $F0
6020 E6 F1      INC   $F1      ;$F0、$F1 指向 $817F
6022 A0 80      LDY   #$80     ;数据块 3 的数据长度置入 Y
6024 20 00 61   JSR   $6100    ;统计数据块 3 的数据类型置入 $FD、
                               $FE、$FF
6027 60         RTS
```

二、数据块的传送

例 7－4 三个数据块分别放在 $8000～$800F、$8100～$810F、$8200～$820F，编程将它们分别顺序搬移到 $6800～$680F、$6900～$690F、$6A00～$6A0F。

关于多个数据块的传送问题，我们在循环程序的设计中，曾以双循环结构进行了圆满解决，现在讨论使用单循环解决这类问题的编程方法。下面先给出程序清单，然后再对个别程序进行说明。程序清单见 No. 7－9。

No. 7－9 多块数据传送程序(一)

```
6000 A9 00      LDA  #$00
6002 85 F0      STA  $F0
6004 85 F2      STA  $F2
6006 A9 90      LDA  #$80      ;第一个源数据块地址
6008 85 F1      STA  $F1       ;$8000 置入 $F0、$F1
600A A9 61      LDA  #$68      ;第一个目的数据地址
600C 85 F3      STA  $F3       ;$6800 置入 $F2、$F3
```

```
600E A9 10     LDA  # $10
6010 85 F4     STA  $F4        ;置送数计数器$F4
6012 20 00 62  JSR  $6200      ;传送第一个数据块
6015 E6 F1     INC  $F1        ;$F0、$F1 指向 $8100
6017 E6 F3     INC  $F3        ;$F2、$F3 指向 $6900
6019 20 00 62  JSR  $6200      ;传送第二个数据块
601C E6 F1     INC  $F1        ;$F0、$F1 指向 $8200
601E E6 F3     INC  $F3        ;$F2、$F3 指向 $6A00
6020 4C 00 62  JMP  $6200      ;传送第三个数据块
```

No. 7－10 数据传送子程序

```
6200 A0 00     LDY  # $00      ;变址计数器 Y 置 0
6202 B1 F0     LDA  ($F0),Y    ;取数
6204 91 F2     STA  ($F2),Y    ;送数
6206 C8        INY             ;变址计数器 Y 加 1
6207 C4 F4     CPY  $F4        ;送完一块数据了吗?
6209 90 F7     BCC  $6202      ;不够转 $6202 继续
620B 60        RTS             ;送完,返回
```

数据传送子程序 No. 7－10 的调用条件是：

（1）$F0、$F1 单元置入传送源数据区首址，$F2、$F3 单元置入传送目的数据区首址，$F4 单元置入数据块长度。

（2）No. 7－10 工作中使用 A、Y 寄存器。

记得我们在阅读程序 No. 6－5 的双重循环结构程序时，比较吃力，把它改成单循环的子程序调用结构，阅读起来就容易多了。由此可见，子程序确实可以起到清晰程序结构的作用。

程序 No. 7－9、No. 7－10 若写成 No. 7－11 的形式，仍不失其灵活性、易读性，而程序的结构更显得紧凑。

No. 7－11 多块数据传送程序(二)

```
6000 A9 00      LDA #$00
6002 85 F0      STA $F0
6004 85 F2      STA $F2
6006 A9 90      LDA #$80        ;第一个源数据块地址
6008 85 F1      STA $F1         ;$8000 置入 $F0、$F1
600A A9 61      LDA #$68        ;第一个目的数据地址
600C 85 F3      STA $F3         ;$6800 置入 $F2、$F3
600E A9 10      LDA #$10
6010 85 F4      STA $F4         ;置送数计数器 $F4
6012 20 20 60   JSR $6020       ;传送第一个数据块
6015 E6 F1      INC $F1         ;$F0、$F1 指向 $8100
6017 E6 F3      INC $F3         ;$F2、$F3 指向 $6900
6019 20 20 60   JSR $6020       ;传送第二个数据块
601C E6 F1      INC $F1         ;$F0、$F1 指向 $8200
601E E6 F3      INC $F3         ;$F2、$F3 指向 $6A00
6020 A0 00      LDY #$00        ;变址计数器 Y 置 0
6022 B1 F0      LDA ($F0),Y     ;取数
6024 91 F2      STA ($F2),Y     ;送数
6026 C8         INY             ;变址计数器 Y 加 1
6027 C4 F4      CPY $F4         ;送完一块数据了吗?
6029 90 F7      BCC $6202       ;不够转 $6202 继续
602B 60         RTS             ;送完,返回
```

程序 No. 7－11 的工作原理留给读者分析。

第八章　堆栈程序设计

在简单程序和分支程序的设计中，我们已经涉及到“堆栈”的应用。实际上，具体编程中很难把各类程序的界限划分的清清楚楚，也没有必要分清哪个问题用堆栈程序结构、哪个问题用子程序结构。往往是从处理问题的实际需要出发，以编程方便为基点，以正确解决问题为宗旨，以执行速度快捷为目标，混合使用各类结构编写程序。我们之所以分为简单、分支、循环等五类程序结构进行讨论，是从方便读者循序渐进地学习角度考虑的。从以上的介绍中也可以看出，子程序中既有分支也有循环，更包含着简单程序结构。因此，本章所讨论的编程方法将是以上所学知识的总结，所罗列的程序将是6527汇编语言程序结构的综合体现。

第一节　现场的保护与恢复

我们知道，6527CPU仅有六个寄存器，而供用户编程使用的寄存器只有A、X、Y三个。因此，在子程序的设计中、特别是在中断服务程序的设计中，现场的保护与恢复就显得极其重要。为了体会现场保护在编程中的必要性，我们先看一个实例。

一、游戏者积分转换程序设计

例8—1 某游戏程序中，两个游戏者的次序号由变址寄

存器 X 指出，X＝0 是 1 号游戏者、X＝1 是 2 号游戏者。已知他们的游戏积分(即累积的分数)依次放在 $81～$83，$91～$93 单元中，每一个单元存放两位十进制数。积分的存放形式如下：假如 1 号游戏者的积分为 216358，各单元的内容为

$81＝21

$82＝63

$83＝58

编写程序，根据 X 的取值，把存在各自三个单元中的 6 位积分，转换为可显示的 6 位 ASCII 码数据，存放在积分送显存储单元 $84～$89 中。

1. 问题分析

(1) $81～$83，$91～$93 单元的操作是相同的，都是把一个字节的数据按字的高、低位顺序，拆分为两个字节，再把它们转换成可显示的 ASCII 码，存放在各个显示存储单元。因此，应考虑编写子程序来完成拆字、组字工作。

(2) 因两个游戏者的积分存放单元不同，要想用一个子程序对不同地址的数据操作，必须采用间接变址寻址方式，本程序选 $8A、$8B 为间接寻址指针。

(3) 转换处理前，应根据游戏者序号寄存器 X 中的内容设置积分取数基地址。当 X＝0 时，基地址为 $0081；当 X＝1 时基地址为 $0091。

(4) 该问题的主程序，实际上是游戏控制程序中的一个子程序。因此，在积分转换操作中，不允许改变 X 寄存器中的内容。也就是说，编程中必须保护 X 寄存器中的内容。

2. 编写转换子程序

(1) 程序清单

假定调用子程序前，已对$8A、$8B单元置入积分单元地址。根据以上思路编写的子程序见No.8－1。

No.8－1 积分转换子程序

```
6100 8A      TXA              ;保护游戏者的序号
6101 48      PHA              ;X 压栈保护
6102 A2 05   LDX # $05        ;置 ASCII 码变址指针初值
6104 A0 02   LDY # $02        ;置取数地址指针初值
6106 B1 8A   LDA ($8A),Y      ;取出一个字节的积分
6108 48      PHA              ;压栈保护
6109 29 0F   AND # $0F
610B 09 30   ORA # $30        ;转换低位
610D 95 84   STA $84,X        ;存后一位 ASCII 码
610F 68      PLA              ;压栈的积分出栈
6110 CA      DEX              ;ASCII 码指针减 1
6111 4A      LSR
6112 4A      LSR
6113 4A      LSR
6114 4A      LSR              ;将高 4 位移入低 4 位
6115 09 30   ORA # $30        ;转换高位
6117 95 84   STA $84,X        ;存前一位 ASCII 码
6119 CA      DEX              ;ASCII 码指针减 1
611A 88      DEY              ;积分字节指针减 1
611B 10 E9   BPL $6106        ;未转换完继续
611D 68      PLA              ;转换完，原压栈保护的 X 出栈
611E AA      TAX              ;恢复游戏者的序号
611F 60      RTS
```

(2) 子程序说明

·因子程序中使用了变址寄存器X作为ASCII码存储地址的变址指针，在子程序执行前对X压栈保护，而在子程

序返回前，又恢复了 X 寄存器的原值。

• 转换操作从积分存储器的最后一个单元向前转换，由 Y 作为取数变址指针。取出一个字节的积分后，首先压栈保存，由 $6109～$610E 程序转换低位；低位处理完毕再令压栈保存的积分值出栈，令 X 减 1，使其指向 ASCII 码存储区的地址前移一个单元；由 $6111～$6118 程序完成高 4 位数据的转换（累加器 A 右移位 4 次，使高 4 位中的内容移入低 4 位）；再令 X 减 1，使 ASCII 码存储区的地址前移一个单元。

• 令积分计数器（积分存储器地址变址指针）减 1，判断若 3 个字节未转换完，转 6106 继续。若转换完则恢复 X 现场，返回调用程序。

3. 编写积分转换控制程序

主程序的功能是设置转换子程序的初始工作状态，即把积分地址基址置入 $8A、$8B 单元。程序清单如 No. 8—2。

No. 8—2 积分转换控制程序

```
6000 A9 81     LDA # $81   ;A=81
6002 E0 00     CPX # $00
6004 F0 02     BEQ $6008   ;X=0 转移
6006 A9 91     LDA # $91   ;A=91
6008 85 8A     STA $8A     ;置游戏者积分地址低位
600A A9 00     LDA # $00
600C 85 8B     STA $8B     ;置游戏者积分地址高位
600E 20 00 61  JSR $6100   ;转换积分
600F 60        RTS
```

程序 No. 8—2 的结构较简单，分析从略。

4. 现场保护分析

现在我们来分析：若没有保护 X 寄存器的内容将会产生

什么样的影响。

在子程序 No. 8－1 中，X 用于转换后的 ASCII 码存储地址变址指针。每存储一个 ASCII 码，X 中的内容减 1，当存储 6 个 ASCII 码后，X 将等于 FFH。这时恰好子程序结束，返回转换控制程序，又由控制程序返回调用程序。这样因 X 中内容的变化，势必引起积分转换以后的程序工作混乱（比如，积分转换后，进行卡通的运动控制处理等），严重时可能因此而死机。

因此，现场保护是十分必要的，在程序设计中也是使用最经常的。当然，上述程序中 X 的保护，也可以在转换控制程序中进行。但要注意一点，在哪一级子程序中进行的压栈保护，必须在哪一级出栈，绝对不允许在一个子程序中压栈而在另一个子程序中出栈。

二、常用现场保护程序的设计

现场保护最彻底、使用最频繁的场合就是中断服务程序。我们知道，游戏程序中只使用了一类中断——非屏蔽中断。而非屏蔽中断的特点就是：中断申请一旦发出，CPU 就立即响应。因此，在主程序处理期间，若遇到中断请求，不管 CPU 正在处理什么样的工作，都将在执行完当前指令后，立即转向中断服务程序。这样在进行中断服务程序处理以前，必须保护主程序的现场，以便中断程序处理结束后予以恢复，使 CPU 接续中断处理前的操作。

常见中断处理程序的现场保护、恢复程序见 No. 8—3。

No. 8－3 中断处理中的现场保护、恢复程序

```
PHA          ;A 压栈
TXA
PHA          ;X 压栈
TYA
PHA          ;Y 压栈
PHP          ;P 压栈
.....
.....        ;中断处理程序
.....
PLP          ;P 出栈
PLA
TAY          ;Y 出栈
PLA
TAX          ;X 出栈
PLA          ;A 出栈
RTI          ;中断返回
```

注意——现场保护中的数据压栈顺序与现场恢复中的数据出栈顺序是相反的，这是由“先进后出、后进先出”的堆栈操作原则决定的，必须牢牢记住，切勿颠倒。

第二节　排序程序的设计

排序问题在数据处理中占有很重要的地位。其目的就是根据某种预先规定的准则，把多个数据按一定的顺序排列起来，如考试成绩排名次、各种竞赛排名次等都要用到排序。

排序有许多分类方法：从数的类型来分有无符号数排序和有符号数排序；从结构上分有从前向后排序和从后向前排序；从要求上分有从小到大递增排序和从大到小递减排序；从方法上又有冒泡排序、交换排序、枚举排序、两极排序等多

种。我们仅以最常用的方法——枚举法、冒泡法为例，介绍排序程序的设计方法。

一、无符号数据排序

例 8－2 无符号数据块放在 $6100～$611F，编程从后向前按递增顺序排序。

1. 编程思路

从后向前按递增顺序排序又称枚举法排序，其操作过程是：

(1) 设数据块的长度为 S，先取第 S 个数据与数据块中第 S－1 个数据比较，若第 S 个数据小，则令两数据交换位置，使较大的数据存入第 S 位；若第 S 个数据大，则取第 S－2 个数据与其比较……。如此一直比较、交换下去，直到第 1 个数据比较、交换后，本轮比较结束，该数据块中第 S 个单元存储的将是该数据块中最大的数据。

(2) 令数据块的总长度 S 减 1(即甩掉已排出的最大数据)再送到 S，重复第(1)项操作，直到排完所有的数据(S＝0)，结束排序操作。

2. 编写程序

根据以上的排序过程编写的程序见 No. 8－4。

No. 8－4 无符号数枚举法排序子程序

```
6000 A0 20     LDY #$20      ;S→Y
6002 84 F6     STY $F6       ;数据块长度送入存入$F6
6004 C6 F6     DEC $F6       ;数据块长度S－1→S
6006 A6 F6     LDX $F6       ;取S→X
6008 B9 FF 60  LDA $60FF,Y   ;取第S个数据
600B DD FF 60  CMP $60FF,X   ;与前面单元的数据比较
```

```
600E B0 0B      BCS $601B       ;第S个数据大转移
6010 48         PHA             ;第S个数据小则压栈
6011 BD FF 60   LDA $60FF,X     ;取较大的数据于A
6014 99 FF 60   STA $60FF,Y     ;较大的数据置入第S位
6017 68         PLA             ;较小的数据出栈
6018 9D FF 60   STA $60FF,X     ;较小的数据送入前面的单元
601B CA         DEX             ;本轮比较的数据指针减1
601C D0 EB      BNE $6008       ;本轮未比较完,继续
601E A4 F6      LDY $F6         ;本轮比较完,取S→Y
6020 C6 F6      DEC $F6         ;数据块长度S-1→S
6022 D0 E4      BNE $6006       ;S≠0,转移继续
6024 60         RTS
```

3. 调用条件

程序 No. 8－4 在调用程序前，必须把排序数据置入 $6100～$611F 单元。

如数据区的数据为：

```
6100 67 D9 34 98 B3 E9 54 12 43 78 B5 C9 E2 42 12 78
6110 00 09 56 C9 A8 B4 D9 E3 E9 DB DA EB AE DE 54 02
```

运行程序后，将把数据区的数据按从小到大顺序排列。

```
6100 EB E9 E9 E3 E2 DE DB DA D9 D9 C9 C9 B5 B4 B3 AE
6110 A8 98 78 78 67 56 54 54 43 42 34 12 12 09 02 00
```

二、有符号数据排序

例 8－3 若例 8－2 数据区的数据为有符号数，编写程序按递增顺序排序。

1. 编程思路

编程仍采用例 8－2 的方法，但因现在参与排序的数据

是有符号数，不能再使用 C 标志进行数据大小的判断。

在比较带符号数据的大小时，可由符号标志位 N、溢出标志位 V 两个标志共同判断。判断方法是：当数据 1(S1)的补码减数据 2(S2)的补码时，若 N、V 两个标志相同(N＝1、V＝1 或 N＝0、V＝0)则 S1≥S2；若 N、V 两个标志不同(N＝1、V＝0 或 N＝1、V＝1)则 S1＜S2。

另外，由于 6527 的比较指令“CMP”操作只影响标志寄存器的 C、N、Z 位，而不影响 V 标志，所以在进行两个有符号数大小比较时只能使用“SBC”指令，不能使用“CMP”指令。

2. 编写程序

根据以上讨论编写的程序见 No. 8－5。

No. 8－5 有符号数枚举法排序子程序

```
6000 A0 20      LDY #$20       ;S→Y
6002 84 F6      STY $F6        ;数据块长度送入存入$F6
6004 C6 F6      DEC $F6        ;数据块长度S－1→S
6006 A6 F6      LDX $F6        ;取S→X
6008 38         SEC            ;置C＝1
6009 B9 FF 60   LDA $60FF,Y    ;取第S个数据
600C 48         PHA            ;压栈，以便进行减运算
600D 9D FF 60   SBC $60FF,X    ;减前面单元的数据
6010 50 0F      BVC $6021      ;V＝0转移查N
6012 30 0F      BMI $6023      ;V＝1、N＝1，不交换，转移
6014 BD FF 60   LDA $60FF,X    ;V＝1、N＝0，交换
6017 99 FF 60   STA $60FF,Y    ;较大的数据送入第S个单元
601A 68         PLA
601B 9D FF 60   STA $60FF,X    ;较小的数据送入前面的单元
601E 4C 24 60   JMP $6024
6021 30 F1      BMI $6014      ;V＝0，N＝1，交换，转$6014
```

```
6023 68        PLA          ;V=0,N=0,不交换,释放堆栈
6024 CA        DEX          ;本轮比较的数据指针减1
6025 D0 E1     BNE $6008    ;本轮未比较完,继续
6027 A4 F6     LDY $F6      ;本轮比较完,取S→Y
6029 C6 F6     DEC $F6      ;数据块长度S-1→S
602B D0 D9     BNE $6006    ;S≠0,转移,继续
602D 60        RTS
```

若执行程序前，$6100～$611F单元的内容仍如例8—2所列。执行程序后，将按数据大小，把负数排在前面、正数排在后面，依次升序排列如下：

```
6100  98 A8 AE B3 B4 B5 C9 C9 D9 D9 DA DB DE E2 E3 E9
6110  E9 EB 00 02 09 12 12 34 42 43 54 54 56 67 78 78
```

注——因负数用补码表示，所以负数的绝对值越大其补码越小。如“－68H”的补码为“98H”、“－15H”的补码为“EBH”。

三、冒泡法排序

1. 冒泡法排序的工作过程

所谓“冒泡”，就是令较大的数据如同气泡一样从数据的底层上升到顶层。冒泡法排序的工作过程是：

（1）取第一、第二两个数据比较，若第二个数据大，则两个数据交换位置，使较大的数据在前面；若第一个数据大，比较结束。

（2）取第二、第三两个数据比较，若后面的数据大，则两个数据交换位置，使较大的数据在前面；若前面的数据大，比较结束。

（3）以后，再取第三、第四两个数据比较，第四、第五两个数据比较，……，一直比较到最后两个数据，一轮比较交换结束。

经过一轮处理后，数据块中最小的数据将移到最后一个单元，较大的数据均不同程度的向前移动。

(4) 下一轮比较重复(1)～(3)的操作。若数据有S个，则比较S－1轮。每进行一轮操作，大数向前移动一位，比较完毕，最大的数将排在第一位、次大的数排在第二位……，最小的数据则排在最后。

显然，冒泡法是按降序进行排序的。

为了加快排序速度，对N个数据来说，一般每比较一轮令S减1(因比较一轮后，最小的数已移入最后一个单元，没有必要再对它进行比较)。即第一轮比较S－1次、第二轮比较S－2次……、第S－1轮只比较1次。

2. 编写排序程序

根据冒泡法排序的工作过程，以无符号数为例编写的排序程序见No. 8－6。

No. 8－6 冒泡法排序程序

```
6000 A0 00   LDY # $00      ;取数变址指针置初值
6002 A6 F6   LDX $F6        ;取排序数据的个数S
6004 F0 22   BEQ $6028      ;数据个数等于0时结束
6006 86 F7   STX $F7        ;预置数据标志非零
6008 B1 F8   LDA ($F8),Y    ;取前面单元的数据
600A C8      INY
600B D1 F8   CMP ($F8),Y    ;与后面单元的数据比较
600D B0 0E   BCS $601D      ;前面的数据大,不交换,转移
600F 48      PHA            ;前面的数据小,交换
6010 B1 F8   LDA ($F8),Y    ;取较大的数据
6012 88      DEY
6013 91 F8   STA ($F8),Y    ;置入前面的单元
6015 68      PLA
```

```
6016 C8      INY
6017 91 F8   STA (＄F8),Y   ;较小的数据置入后面的单元
6019 A9 00   LDA ＃＄00
601B 85 F7   STA ＄F7       ;置数据交换标志为零
601D CA      DEX            ;数据比较次数减1
601E D0 E8   BNE ＄6008     ;未到S－1次,继续
6020 A5 F7   LDA ＄F7       ;本轮比较结束，查交换标志
6022 D0 04   BNE ＄6028     ;没有交换,排序结束
6024 C6 F6   DEC ＄F6       ;有交换,令S－1→S
6026 D0 D8   BNE ＄6000     ;继续
6028 60      RTS
```

3. 程序说明

（1）本程序使用了间接寻址方式，基址存储器为＄F8、F9，排序前应将数据区首址置入＄F8、＄F9，将数据区长度置入＄F6单元。

（2）＄6008～＄600E为取数、比较、判断程序，若前面的数据大则不交换；若后面的数据大，则进行数据交换处理。

（3）＄600F～＄601C为数据交换处理程序，首先把较小的数据压栈，然后把较大的数据置入前面的单元，最后压栈的数据出栈，置入后面的单元。前、后单元地址变更由变址寄存器Y控制。

（4）为控制排序完成后及时退出循环，程序中设置了数据交换标志单元＄F7。

＄F7的控制作用如下：每轮比较前，置＄F7初值为非零值，比较中若各数据均按大小顺序排列完毕，必然没有数据交换，＄F7将保持其非零状态。这样，在本轮比较结束后，将由＄6022程序控制转出排序处理。反之，若发生数据交换时，必有＄F7等于0，因此将控制比较继续进行下去。

（5）本程序有一个缺点是：执行最后一次交换，排序完成后，因置 $F7 等于 0，还要多循环一次。

（6）$6004 指令的作用是拦截排序数据个数为 0 的情况，以避免进行错误排序，也可以省略不用。

4. 排序结果的检查

程序 No. 8—6 编写完成后，可以使用 F BASIC 语言编写一个随机数产生程序，把随机数置入 RAM 单元，然后，调用 No. 8—6 进行排序。排序结束后，打印出排序结果。BASIC 程序见 No. 8—7。

```
10 REM No. 8—7
20 PRINT "SUI JI SHU"
30 FOR I=0 TO 127
40 A=RND (200): PRINT A;" ";
50 POKE &6100+I,A
60 NEXT: PRINT
70 PRINT "PAI XU HOU"
80 POKE &F6,127,0,0,&61
90 CALL &6000
100 FOR I=&6100 TO &617F
110 PRINT HEX$ (PEEK(I));" ";
120 NEXT
```

程序说明：

20～60 行产生、打印随机数，并把数据置入 RAM 区 $6100～$617F 单元；

80 行向 $F6 单元置入数据区长度、向 $F8、F9 单元置入数据区首址 $6100；

90 行调用排序程序，完成排序；

100～120 行打印排序后的数据供检查。

第三节　乘除运算程序的设计

关于加、减运算程序设计，已在简单程序设计中进行了介绍。本节仅以 BS·2A 版本 F BASIC 系统程序中的乘、除运算子程序为例，介绍 16 位带符号二进制数乘、除运算程序的设计方法。

一、乘法运算程序设计

1. 乘法运算程序的组成

乘法运算子程序为两字节(16 位二进制数)的带符号整数运算程序，计算数值范围为－32768～＋32767(－8000H～＋7FFFH)。处理程序包括：

“两字节乘法运算”子程序 No. 8－8

“两因数正、负判断，补码转换，设置乘积符号标志”子程序 No. 8－9

“乘数补码、原码转换”子程序 No. 8－10

“被乘数补码、原码转换”子程序 No. 8－11

注——上述程序进行了一定的化简处理。

在调用乘法运算子程序之前，须把被乘数置入 $28、$29 单元，乘数置入 $2C、$2D 单元。乘法运算结束后，乘积存在 $28、$29 单元。

2. 两字节乘法运算子程序清单

No. 8－8 两字节乘法运算子程序

```
8B56 A900     LDA   # $00
8B58 A208     LDX   # $08
8B5A 9534     STA   $34,X
```

```
8B5C CA      DEX
8B5D D0FB    BNE   $8B4B      ;令装积单元$35～$3C=0
8B5F 20928B  JSR   $8B92      ;两因数正、负处理
8B62 48      PHA               ;乘积符号标志压栈
8B63 A210    LDX   #$10  ;┐
8B65 18      CLC              |
8B66 6629    ROR   $29        |
8B68 6628    ROR   $28        |
8B6A 900D    BCC   $8B79      |
8B6C 18      CLC              |
8B6D A536    LDA   $36        |
8B6F 652C    ADC   $2C        |
8B71 8536    STA   $36        |  循环右移位
8B73 A537    LDA   $37        |  进行乘法运算
8B75 652D    ADC   $2D        |
8B77 8537    STA   $37        |
8B79 6637    ROR   $37        |
8B7B 6636    ROR   $36        |
8B7D 6635    ROR   $35        |
8B7F 6634    ROR   $34        |
8B81 CA      DEX              |
8B82 D0E2    BNE   $8B66 ;┘
8B84 A534    LDA   $34
8B86 8528    STA   $28        ;存乘积的低位
8B88 A535    LDA   $35
8B8A 8529    STA   $29        ;存乘积的高位
8B8C 68      PLA               ;乘积符号出栈
8B8D F022    BEQ   $8BB1      ;乘积符号为正,子程序返回
8B8F 4C7C8A  JMP   $8A7C      ;乘积符号为负,转换为补码
```

No. 8—9 两因数正、负判断,补码转换,

设置乘积符号标志子程序

```
8B92  A900     LDA   # $ 00
8B94  8534     STA    $ 34      ;预置符号标志 $ 34=0
8B96  A529     LDA    $ 29      ;判断被乘数
8B98  1007     BPL    $ 8BA1    ;被乘数是正数转移
8B9A  207C8A   JSR    $ 8A7C    ;被乘数是负数,还原绝对值
8B9D  A9FF     LDA   # $ FF
8B9F  8534     STA    $ 34      ;置负数标志 $ 34=FF
8BA1  A52D     LDA    $ 2D      ;判断乘数
8BA3  100A     BPL    $ 8BAF    ;乘数是正数转移
8BA5  20618A   JSR    $ 8A61    ;乘数是负数,还原绝对值
8BA8  A9FF     LDA   # $ FF
8BAA  38       SEC
8BAB  E534     SBC    $ 34      ;两数同号,置 $ 34=0
8BAD  8534     STA    $ 34      ;两数异号,置 $ 34=FF
8BAF  A534     LDA    $ 34      ;取乘积的符号标志于 A,返回
8BB1  60       RTS
```

No. 8—10 补码——原码转换子程序(一)

```
8A61   38      SEC
8A62   A900    LDA   # $ 00
8A64  E52C     SBC    $ 2C
8A66  852C     STA    $ 2C
8A68  A900     LDA   # $ 00
8A6A  E52D     SBC    $ 2D
8A6C  852D     STA    $ 2D
8A6E  60       RTS
```

No. 8—11 补码——原码转换子程序(二)

```
8A7C  38       SEC
8A7D  A900     LDA   # $ 00
8A7F  E528     SBC    $ 28
```

```
8A81  8528    STA   $28
 A83  A900    LDA   #$00
 A85  E529    SBC   $29
 A87  8529    STA   $29
8A89  60      RTS
```

3. 乘法运算子程序工作过程

(1) 乘法运算前先对工作单元 $34～$3C 清零。

(2) 调 $8B92 子程序，根据被乘数、乘数的符号确定乘积的符号存入 $34 单元，若两数符号相同(都为正或都为负)，则置 $34＝0；若两数符号不同则置 $34＝FF。

(3) 在调 $8B92 中，若被乘数为负数则调子程序 No. 8－11，把以补码形式表示的被乘数还原为原码仍存于 $28、$29 单元；若乘数为负数则调子程序 No. 8－10，把乘数还原为原码仍存于 $2C、$2D 单元。这里的两个补码、原码转换子程序可以实现数据的补码、原码互相转换，即补码可以转换为原码，原码可以转换为补码。请读者注意分析这两个子程序的编程方法与技巧。

(4) 执行 $8B62 指令，把乘积符号标志压栈。

(5) 执行 $8B63～$8B83 程序，进行循环右移位乘法运算(请读者自己分析运算过程)。运算结束后，$34 单元存乘积的低位、$35 单元存乘积的高位。

(6) 执行 $8B84～$8B8B 程序，把乘积转存到 $28、$29 单元。

(7)乘积符号标志出栈，若是正数则直接返回；若是负数则转子程序 No. 8－11，把乘积转换为补码仍存于 $28、$29 单元，结束乘法运算。

二、除法运算程序的设计

系统除法运算子程序的处理方式、顺序与乘法基本相同，被除数存于 $28、$29 单元，除数存于 $2C、$2D 单元，运算结束后，商存在 $28、$29 单元。

除法运算前，也要调程序 No. 8－9 进行两数的负数处理和商的符号判断；然后以循环左移位的方式完成除法运算；运算结束后转 $8B8C 程序段，把商转存到 $28、$29 单元，并根据商的符号决定是否进行补码转换。

除法子程序见 No. 8－12，程序分析留给读者练习。

No. 8－12 除法运算子程序

```
8BB5 20928B   JSR $8B92    ;两数符号的判断处理
8BB8 48       PHA          ;商的符号标志压栈
8BB9 A900     LDA #$00
8BBB 852A     STA $2A
8BBD 852B     STA $2B
8BBF A210     LDX #$10     ;┐
8BC1 0628     ASL $28      ;│
8BC3 2629     ROL $29      ;│
8BC5 262A     ROL $2A      ;│ 循环不左移位
8BC7 262B     ROL $2B      ;│ 进行除法运算
8BC9 A52A     LDA $2A      ;│
8BCB 38       SEC          ;│
8BCC E52C     SBC $2C      ;│
8BCE A52B     LDA $2B      ;┘
```

```
8BD0 E52D    SBC  $ 2D
8BD2 900F    BCC  $ 8BE3
8BD4 A52A    LDA  $ 2A
8BD6 38      SEC
8BD7 E52C    SBC  $ 2C
8BD9 852A    STA  $ 2A      循环左移位
8BDB A52B    LDA  $ 2B      进行除法运算
8BDD E52D    SBC  $ 2D
8BDF 852B    STA  $ 2B
8BE1 E628    INC  $ 28
8BE3 CA      DEX
8BE4 D0DB    BNE  $ 8BC1 ;
8BE6 4C8C8B  JMP  $ 8B8C ;转移,进行商的处理
```

总之，堆栈程序设计的用途很广泛，数据的压栈、出栈操作也很频繁，使用中，请读者注意以下两点：

第一，堆栈操作的原则是“先进后出、后进先出”，压栈和出栈的顺序是相反的；

第二、数据的压栈、出栈操作必须在一个子程序内完成，初学者不要进行“在这一个子程序中压栈，而在另一个子程序中出栈”的尝试，以避免发生死机。

第九章　系统资源的利用

BS·2A 版本 F BASIC 工作系统的解释程序占 32K 字节，其中有十余类、百余个独立的功能子程序可供我们在编程中调用，动态演示八皇后程序中的背景音乐伴奏就是调用系统音乐演奏程序的典型实例。

正确、灵活地调用系统子程序，可使应用程序量得到极大地压缩，可以较小的 RAM 资源实现较强的功能，编写出性能优异的应用软件。笔者曾使用 3K 字节编写出具有“区位、全拼、双拼、五笔”四种汉字输入法、磁带存取文件、打印文件功能的全屏幕汉字文字处理软件。这足以说明利用系统子程序的优势。

我们知道，零页寻址方式指令，占用字节少、执行速度快，大家都喜欢使用零页存储器编写程序。但由于我们的汇编程序是在 F BASIC 工作环境下建立的，零页单元的使用必须避免与系统的冲突，故了解系统对零页乃至其它页 RAM 的使用情况，对编写机器语言程序具有不可估量的价值。

本章将向读者介绍 F BASIC 工作系统对零页单元的使用情况，使大家掌握在什么情况下可以使用零页的哪些单元，以做到灵活运用、互不干扰。还将介绍可供调用的系统子程序入口以及调用条件，使大家做到心中有数，合理调用。最后一节将给出一个与 F BASIC 系统并行工作的电子钟程序，作为调用系统子程序的实例，供大家参考。

第一节　系统对 RAM 的使用

F BASIC 工作系统把 RAM 区进行了仔细的、确切的分配。特别是对 $0000～$07FF 这 2K 字节分工明确、不许误用。一旦误用，轻则造成运算结果错误；重则造成死机。只有全面掌握 RAM 中哪些单元禁止使用、哪些单元可以有条件使用、哪些单元可自由使用，才能保证机器语言程序既能顺利运行又不影响系统工作。

下面列出系统对 RAM 单元的使用情况，凡前面有双星号(“**”)的单元，禁止使用；前面有单星号(“*”)的单元谨慎使用；没有星号的单元可以有条件地使用；没有列出的单元可自由使用(有“* *”号的为特殊单元)。

一、零页单元的分工

* $00：进入 F BASIC 工作系统的标志
  $02：PALETB 指令中的计数器
* $03、$04：RAM 自由区末址
* $05、$06：用户程序区首址
* $07、$08：用户程序区末址
* $09、$0A：执行程序的变址指针
* $0B、$0C：当前执行程序行的行号
* $0D、$0E：下一行程序在内存中的首址
* $0F、$10：翻译程序时暂存程序行的行号
  $11、$12：STOP 中暂存当前执行程序行的行号
  $13、$14：STOP 中暂存下一程序行首址
  $15、$16：STOP 中暂存当前执行指令的地址

* $17:程序转向前的栈指针

* $18:读键工作单元。按 Ctrl、Alt、Shift 键时 $18=FF,否则 $18=0。

* $19、$1A:执行程序时,暂存将被执行指令的入口地址或字符串值地址,SWAP 中存第一变量名。

* $1B、$1C:执行程序中暂存数据,在 SAVE 中作计数器;在系列中断处理程序中为转移程序入口;$1B 在 READ、INPUT 中存字符串的长度。

* $1D、$1E:BASIC 程序变量区首址

* $1F、$20:BASIC 程序变量区末址

* $21、$22:RAM 自由区首址。$21 单元还作为七页变址寻址器。

* $24:CONT 指令有效标志。有效时置 FFH;无效时置 0。

* $25、$26:POKE 中暂存写入地址

* $27:暂存变量类型标志

* $28～$2A:第一操作数,$2A 为操作数的种类。在 IF 语句中,存程序运行中对变量的赋值。

* $2C～$2E:第二操作数,$2E 为操作数的种类。在 IF 语句中存变量在变量区的数值。

* $30:四页自由区首址

* * $31:屏幕显示标志单元,记录 $2001 软开关的参数。

* * $32:显示页面对图形库的选择标志单元

* * $33:屏幕显示状态存储单元

$34～$3B:乘、除运算工作单元

$3C～$42:运算结果暂存单元。对于十进制数结

果 $42=0，从 $3D～$41 依次存万、千、百、十、个位，$3C 为符号位。对于十六进制数 $41=0，$3D～$40 存四位数。

* $44、$45：程序转向时存目标行号，翻译程序时存行号。

$46：暂存送入 $3D 及以后单元的运算结果的 ASCII 码位置。还作为 POKE 的变址指针。

$47：暂存送入 $3D 及以后单元的运算结果的 ASCII 码。

按回车键后，暂存本次输入的首行 Y 座标。

* * $48：光标的当前水平座标

* * $49：光标的当前垂直座标

* $4A、$4B：SPRITE、COLOR 中暂存 X、Y 参数的 PPU 地址。系列中断程序中为 PPU 的地址指针。

* $4C：显示页配色区状态标志

$4D、$4E：程序翻译中作为三页的地址指针

* * $4F：背景显示页面在 PPU 中的起始地址高位。00 页为 H20、10 页为 H24。

$50：Capslock 键工作单元

$52、$53：在间接变址中用于设置转移入口。INPUT 中 $52 用于记录输入字符存于五页的首址。

$54、$55：程序翻译中用于变址寻址

$56、$57：随机数发生器

$58、$59：执行程序时，暂存当前处理指令的地址。

$5A：三页自由区指针。程序翻译时存本语句的长

度。

* $5B、$5C：暂存执行程序行的地址

$5D：检查动作卡通状态的计数器

$5F、$60：在执行循环语句、转子语句时暂存栈顶两单元的内容。

* * $61：光标形状存储单元

$62：七页地址指针

* * $63：系统中断处理转向控制器

$64：七页送显的首址指针

$65：PALET 标志。PALETB 时 $65 = 0、PALETS 时 $65=1。

* $66：功能键输入标志。按功能键时 $66=FF。

* $67：功能键字符地址指针。高位为功能键编号、低位为字符个数。

* $68：读完功能键和空键标志。读完时 $68=FF。

* * $69：屏幕滚动控制开关。可上、下移动整幅画面。

$6A：读功能键时暂存键值

$6B、$6C：暂存 LOCATE 指令中的 X、Y 参数

* $6D、$6E：读键中，键值与键码换算工作单元。

* $6F：Ctrl 键工作单元。按该键时 $6F=FF。

* $70：Shift 键工作单元。按该键时 $70=FF。

* $71：Alt 键工作单元。按该键时 $71=FF；

* $72：Capslock 键工作单元。按一次该键时 $72=FF；再按一次 $72=0。

* $73：读 PPU 工作单元

$74：计数器

* $75：键码暂存器

* $76:五页地址指针

$77～$79:连读键功能建立工作单元。其中 $79 为稳键时间控制器。

$7B:连读键标志。连读时 $7B=FF。

$7C、$7D:DIM 时暂存数组名称。循环语句中暂存循环变量值在变量区的首址。

$7E、$7F:DIM 时暂存数组的两个下标

$80、$81:暂存循环变量的终值

$82、$83:暂存循环步长。DIM 时,$82 存数组维数标志,一维置零,两维置一。

$84、$85:存循环语句的下一行首址。$84 还作为变量类型暂存器,简单变量置 2;串变量置 3;数型下标变量置 82H;串型下标变量置 83H。

$86、$87:暂存循环语句的行号。LIST 时存起始行号。

$88、$89:暂存循环体首址。LIST 时存终止行号。

$8A、$8B:变量名称暂存器

$8C、$8D:暂存当前变量值的存放地址

$8E、$8F:可使用变量区首址

$90:变量类型标志。KEY 时作为字符计数器;STICK 、STRIG 时暂存其函数值。

$91、$92:SAVE 中作计数器。$91 还作为 PLAY 中的变址指针。

$98～$9B:程序翻译工作单元。其中 $9A、$9B 还作为五页地址指针和 $09、$0A 地址指针的暂存器。

$9C:COLOR 语句中的配色代码

* * $9D:画面横向座标控制器,可左、右移动整幅画面。

$9E：背景一页禁清标志。

$9F：执行 LOAD 标志。LOAD 时置 0，LOAD? 时置 FFH。

$A4：存动作卡通的编号。

$A5：DEF MOVE 工作单元，暂存卡通块编号的八倍值。

$A6～$A8：动作卡通工作单元

$A9：中断处理中检查动作卡通状态时存卡通编号。

$AB：动作卡通启动标志

$AC：MOVE n 时，置入 FFH。

$AD、$AE：SAVE 中存五页的起始地址。$AD 还作为向 PPU 写入控制器。定义卡通时暂存卡通的符号合成型参数。

$AF、$B0：SAVE 中存五页的结束地址。

$B1：LOAD 工作单元

$B2、$B3：程序转向时存目标行的程序体首址。读 DATA 语句时存数据地址。

$B4：读到 DATA 语句标志，读到时置 $B4=FF。

$B5：SAVE、LOAD 中的文件名标志，有文件名时置“1”。

$B6、$B8：出错语句的行号。

$B9～$D4：PLAY 工作单元。其中 $B9～$BB 存三个系统的节拍参数；$BC～$BE 存 V 参数；$BF～$C1 存 O 参数；$C3～$C4 存音长参数；$C6～$C7 存三个系统的弦乐资料在七页的存储地址；$C9 为演奏标志；$CB～$CD 存三个系统演奏有效标志；$CF～$D1 存 M 参数；$D2～$D4 存 Y 参数。

$D5～$DD：动作卡通工作单元。

* $E0：光标形状(常态)。

* $E1：光标形状(小写字母状态)。

$E4：背景零页标志。

$E5：背景一页标志。

* * $ED～$EF：非屏蔽中断入口。

$F0：显示页面标志。

$F1～$FF：背景音乐、效果音响工作单元。

二、系统一页单元的分工

$100～$1FF：为堆栈工作区。

三、系统二页单元的分工

卡通定义区。其中 $200～$27F 为显示卡通定义区；$280～$2FF 为动作卡通定义区。

四、系统三页单元的分工

$300～$3FF：为命令、程序翻译后的存储区。

五、系统四页单元的分工

$400～$47F：工作中间数据暂存区。

$480～$4FF：功能键定义区。每个功能键占连续的 16 个字节，共定义 F1～F8 八个功能键。

六、系统五页单元的分工

$500～$5FF：键盘输入缓存区。

七、系统六页单元的分工

$600、$619：背景零页使用标志区的开始与结束单元。

$601～$618：背景零页 24 行的使用标志单元。当该行占满时对应单元置入 FFH；否则，置入零。

$61A、$633：背景一页使用标志区的开始与结束单元。

$61B～$632：背景一页 24 行的使用标志单元。当该行占满时对应单元置入 FFH；否则，置入零。

$63D～$644：键盘扫描中的键值暂存区。

$645～$64C：存八个动作卡通的状态。

$680～$69F：存八个显示卡通的 32 块卡通块的序号。

$6C0～$6FF：存八个动作卡通的定义数据，每个卡通占八个单元。

八、系统七页单元的分工

$700～$7FF：屏幕显示缓冲区和中间数据暂存区。其中 $700～$763 为 PLAY 语句工作单元，用于暂存三个系统的弦乐资料。

九、用户程序储存区 $6000～$7FFF。

$6000～$6FFF：空区。

$7000～$703D：系统标记区。

$703E～$7FFF：F BASIC 程序存储区。

以上凡带有双星号的单元禁止使用；带单星号的单元在作好现场保护后可以使用。

第二节　系统子程序简介

一、移位类子程序

1. EB3F

功能：令 $8E～$9F 的内容循环前移一个单元。

注意：保护 $00、$01、X、Y。

2. EB34

功能：令 $8E～$9F 的内容循环后移一个单元。注意：保护 $00、$01、X、Y。

3. EB76

功能：令连续的八个单元的内容循环前移一个单元。

注意：调用前要对 $00～$01 置入移位单元的首址；保护 X、Y。

4. EB87

功能：令连续的 16 个单元的内容前移一个单元。

注意：同 3 。

5. EB98

功能：令连续的八个单元的内容后移一个单元。

注意：同 3 。

6. EBAF

功能：令连续的 16 个单元的内容后移一个单元。

注意：同 3 。

二、数制转换类子程序

1. 8CF9

功能：把用 ASCII 码表示的十进制数转换成十六进制数值存于 $28、$29 单元中。数据地址由 $09、$0A 指出。

注意：数据的 ASCII 码要预先存入 RAM 区，并以 0 为结束标志，令 $09、$0A 指向数据首址。

2. 8D3C

功能：把存于 $28、$29 单元的十六进制数据转换成十进制数的 ASCII 码送入 $3D～$41 中，以便输出。

注意：把数据预先置入 $28、$29 单元。

3. 8DD8

功能：把 $28、$29 中的 16 进制数据转换为 16 进制数的 ASCII 码存 入 $3D～$40 单元，令 $41＝0 。

4. 8DEF

功能：把 ASCII 码形式的 16 进制数还原为 16 进制数值存入 $28、$29 中。

注意：要求 $09、$0A 指向数据存储区首址，并以 0 为数据结束标志。

5. 8CE9

功能：对存入 $3D～$41 中待输出的数据整形，删除数据前的 0 。

三、置数、送数类子程序

1. 8B47

功能：令 $35～$3C＝0

2. A1CD

功能：令 $29＝0 、$2D＝0 。

3. A2BA

功能：令 $28＝$29＝$2A＝0 。

4. EC4C

功能：令 $8E～$9F＝0 。

5. DF8F

功能：令 $7600～$76FF＝0 。

注意：保护 A、Y 。

6. A6DF

功能：令 $200～$2FF＝EF 。

注意：保护 A、X 。

7. 98A1

功能：令 $580～$58F＝20 、$590＝0 。

注意：保护 X、Y、A。

8. 8A8D

功能：把 $2C、$2D、$2E 的内容送入 $28、$29、$2A 单元中。

9. 8FF8

功能：保存 $09、$0A 的地址指针于 $9A、$9B 中。($09)→$9A、($0A)→$9B 。

10. 9001

功能：送还 $09、$0A 的地址指针。($9A)→$09、($9B)→$0A 。

11. A0B0

功能：($28)→$8C 、($29)→$8D

12. A007

功能：($8E)→$8C 、($8F)→$8D

13. 98B4

功能：把($580～$58F)→ $501～$510，并令 $511＝0 。

注意：保护 X、Y、A 。

14. DF9A

功能：把 $63D～$714 的内容送入 $7100～$71D7 。

注意：保护 A、X 。

四、运算类子程序

1. 8B0D

功能：令 $28、$29 的内容与 $2C、$2D 相减，差存于 $28、$29 中。

过程：C＝1、($28)－($2C)→ $28、($29)－($2D)→$29

2. 8B39

功能：令 $28、$29 的内容与 $2C、$2D 相加，和存于 $28、$29 。

过程：C＝0、($28)＋($2C)→$28、($29)＋($2D)→$29。

3. 8BB5

功能：令($28、$29)÷($2C、$2D)，商存于 $28、$29 。

注意：保护 $28～$2D、$34、A、X、P 。

4. 8B5C

功能：令($28、$29)×($2C、$2D)，积存于 $28、$29 。

注意：保护 $28～$2D、$34～$3C、A、X、P 。

5. 8A7C

功能：对 $28、$29 单元的内容求补码，结果仍存 $28、$29 中。

过程：C＝1 、0－($28)→$28、0－($29)→$29 。

6. 8A61

功能：对 $2C、$2D 单元中的内容求补码，结果仍存于 $2C、$2D 中 。

过程：C＝1、0－（$2C）→ $2C、0－（$2D）→ $2D 。

7. 8BF7

功能：比较 $29、$28 与 $2D、$2C 的大小。

8. 8C09

功能：比较 $28、$29 与 $2C、$2D 的大小：若（$28、$29）＝（$2C、$2D）则令 C＝1、A＝0 ，返回；若（$28、$29）＞（$2C、$2D）则令 C＝1、A＝FF，返回；若（$28、$29）＜（$2C、$2D）则令 C＝0、A＝FF，返回。

9. 8CDD

功能：比较 A 寄存器中的数据：若等于 30～39 则令 C＝0，返回；否则令 C＝1 返回。

10. 98CC

功能：比较 $501～$511 与 $580～$590 的内容，若相同则令 C＝ 0 返回；若不同则令 C＝1 返回。

注意：保护 A、X、Y。

11. 8E22

功能：令 $28、$29 中的数据扩大 16 倍。

注意：保护乘法运算的现场。

12. A341

功能：令 A 的内容乘 8 后，再送入 X 。

13. A6C5

功能：令 A 中的内容连续左移位四次。

14. EDE8

功能：随机数发生器，随机数存于 $51～$58 单元中。

注意：保护 $00 单元。调用前要先对 $51 单元置入数值。

五、变址取数类子程序

1. 8CC7

功能：读数 A＝($09)＋ Y

说明：$0A、$09 为地址指针，$0A 存高位、$09 存低位。

2. 8C9D

功能：读数 A＝($09)＋ Y ，$09＋1 ，若 $09＝0 则 $0A＋1 。

3. 8CD1

功能：先令 $09＋1，再读数。A＝($09)＋ Y ，若 A＝20 则令 $09＋1 后再读。

4. 8375

功能：先令 $09＋1，再读数。A＝($09)＋ Y，并令 A 与 3AH 比较。

5. 8E35

功能：读数 A＝($4D)＋ Y 。

说明：$4E、$4D 为地址指针，$4E 存高位、$4D 存低位。

6. 8E32

功能：先令 $4D＋1，再读数。A＝($4D)＋ Y 。

7. 8FA2

功能：读数 A＝($54)＋ Y 。

说明：$55、$54 为地址指针，$55 存高位、$54 存低位。

六、地址指针设置类子程序

1. 8522

功能：令 $19、$1A 地址指针减一，若 $19=0 则令 $19=FF、$1A－1。

2. 8516

功能：令 $0B、$0C 地址指针减一，若 $0B＝0 则令 $0B=FF、$0C－1。

3. 8CA1

功能：令 $09、$0A 地址指针加一，若 $09＝0 则令 $0A+1 。

4. 9927

功能：令 $09、$0A 地址指针加三，若有进位则令 $0A+1 。

5. 8FA7

功能：令 $54、$55 地址指针加一，若 $54=0 则令 $55+1 。

6. B3B2

功能：令 $91、$92 地址指针加一，若 $91=0 则令 $92+1 。

7. B3AB

功能：令 $AD、$AE 地址指针加一，若 $AD＝0 则令 $AE+1 。

七、字符、图形显示类子程序

1. 87F8

功能：以 $52、$53 为显示内容的数据地址指针，以 Y

变址，先 把数据读出送入七页，然后再令 A＝1 → $63，以显示字符、图形。

条件：. 向 $52、$53 单元置入送显数据区首址，低位在前；

. 向 $62 单元置入使用七页的首址；

. 向 $48、$49 单元置入显示内容的屏幕初始座标。

2. B254

功能：令光标显示。

注意：作好乘法现场保护。

3. B25B

功能：令光标隐藏。

注意：同 2 。

4. AE48

功能：在光标处挪出一个空格，相当于按 Ins 键。

注意：作好乘法现场保护，还使用 $6B、$6C、$B0、A、X、Y 。

5. ADB2

功能：删去光标前的一个空格，相当于按 Del 键。

注意：同 4，还使用 $19、$1A 。

6. ADA1

功能：改变光标形状。

要求：调用前把光标形状的代码存入 $E0、$E1 单元。

7. AD8B

功能：置光标于屏幕左上角。

8. AD76

功能：令光标右移一格，相当于按 → 键。

9. AD64

功能：令光标左移一格，相当于按 ← 键。

10. AD5A

功能：令光标下移一行，相当于按 ↓ 键。

11. AD53

功能：令光标上移一行，相当于按 ↑ 键。

12. ACF4

功能：清除光标后的所有显示内容，相当于按 Ctrl＋Z 键。

注意：保护乘法现场。

13. ACA1

功能：保存光标的当前座标值于 $6B、$6C。

过程：($48)→$6B、($49)→$6C

14. ACAA

功能：从 $6B、$6C 取回光标的原座标值。

过程：($6B)→$48、($6C)→$49

15. AC45

功能：令光标在原位下移一行，若处于最底行则屏幕上滚一行。

注意：保护乘法现场。

16. AC07

功能：令光标下移一行，位于行首。

17. ABA6

功能：根据光标的当前座标 $48、$49 的数值，计算其对应的 PPU 地址存入 $4A、$4B。

注意：保护乘法现场。

18. 87F8

功能：把以 $52、$53 为地址指针指向的 RAM 区数据

取出送入七页 显示。

19. 87EE

功能：当向七页送数结束后，再送一个零作为结束标志。

过程：X＝＄62、A＝0、A → ＄700＋X、＄62＋1

20. 87F0

功能：把 A 寄存器中的内容送入七页。

21. 87B7

功能：以 Y＝0 变址，取出＄52、＄53 指向 RAM 区的内容送入七页，七页以＄62 变址。

22. 8761

功能：送显字符串。

要求：. 字符串长度存入＄400

. 字符串首址存入＄401、＄402，低位在前

23. 865F

功能：令＄52、＄53 指向＄3D～＄41 中不为零的单元，然后把＄3D～ ＄41 中的内容送入七页显示。

24. ABFA

功能：中断开始后开显示

25. ABF1

功能：中断开始后关显示

26. ABFD

功能：开显示

27. ABF4

功能：关显示

28. A7C0

功能：开卡通页显示

29. A7CC

功能：关卡通页显示

30. A9FF

功能：送显“OK”。

31. AC34

功能：送显 A 寄存器中的内容

32. AC07

功能：令光标下移一行，位于行首。

八、与 PPU 通讯类子程序

1. AAFE

功能：写 PPU 前的处理

2. ABE6

功能：写 PPU 后的处理

3. ABD8

功能：设定 PPU 通讯地址

要求：把 PPU 地址高位置入 $4B、低位置入 $4A 。

4. 81F8

功能：设定 PPU 通讯地址

要求：把 PPU 地址高位置入 A、低位置入 X

5. 8876

功能：读出由 $4A、$4B 指出的 PPU 地址内容的 ASCII 码存入 $73 单元。

6. 819C

功能：在屏幕上画一个大方框。

7. B3F1

功能：开中断

8. B3EB

功能：关中断

9. EE43

功能：查表法绘图(向 PPU 送数)

要求：数据区首址存于 $00、$01 中，低位在前。数据区中第一、第二数据为 PPU 首址，高位在前。

九、清屏延时类子程序

1. AB75

功能：清背景零页

2. AB71

功能：清背景一页

3. 91B3

功能：延时

要求：调用前要对 $400、$401 置入延时参数。$400 置 1，相当于 PAUSE1；$401 置 1，相当于 PAUSE 256。

4. 86B9

功能：扫描 STOP 键，可作为短延时子程序调用。

十、发声类子程序

1. B314

功能：发“嗒”声，(按键声)。

2. B30F

功能：蜂鸣声

3. FD5B

功能：用第一主声部发出四种效果音。

说明：对 $FF 置入 2、4、8、40

4. FE14

功能：用第二主声部演奏背景音乐。

说明：对$FC 置入 1、2、3

5. FE93

功能：用第三主声部发出音乐片断。

说明：对$FE 置入数据。

十一、中断返回类子程序

1. 8862

功能：开背景页显示

2. 885F

功能：关背景页显示

3. 8990

功能：把存入七页的背景配色数据送入$3F××

过程：3FH→$2006、($700)×4→$2006、($701)→$2007、……($704)→$2007；($9D)、($69)→$2005；($32)OR 80H→$2000；$63=0；($9D)、($69)→$2005

4. 89BE

功能：把存入七页的卡通配色数据送入$3F××

过程：同上

5. 89CC

功能：执行 Ins 操作。

过程：($4B)、($4A)→$2006；A=($2007)→$500+X；($4B)、($4A)→$2006；A=($500,X)→$2007；($9D)、($69)→$2005；0→$63

6. 8A02

功能：从$4B、$4A 指向的 PPU 地址开始，把连续的1CH 个显示单 元的数据读出存入$500+X ，最后令$63=

0F。

7. 8A1A

功能：令 $49+1，从 $500+X 中取数送入 $4B、$4A 指向的 PPU 显示 单元中，连送 1CH 个

8. 8961

功能：取出 $4B、$4A 指向 PPU 单元的内容送入五页（共取光标开始到行尾的内容）；把五页的内容送入由 $1A、$19 指向的 PPU 单元（数量与取出时相同）。

过程：($4B)、($4A)→$2006；A=($2007)→$500+X；($1A)、($19)→$2006；A=($500+X)→$2007。取数、送数的起始位置由 $48 决定。

9. 8925

功能：取出与 $49 对应的一行内容存入 $500 以后的单元中。

10. 88AA

功能：取出由 $64 指定的七页内容送入当前光标对应的 PPU 单元中。

说明：当达到右边界时，令 $48=0、$49+1；

如($49)<18H 则中断返回；

如($49)=18H 则令显示上卷一行。

11. 8898

功能：清除从光标开始到行尾的全部内容。

12. 8890

功能：把 $73 单元的内容送入由 $4B、$4A 指向的 PPU 中。

13. 8882

功能：读出由 $4B、$4A 指定 PPU 的内容存入 $73 单

元；把光标形 状(＄61)送入 PPU。

过程：(＄4B)、(＄4A)→＄2006；A＝(＄2007)→＄73；(＄4B)、(＄4A)→＄2006；(＄61)→＄2007

十二、读键盘、操纵器类

1．B343

功能：读出当前键盘输入的内容，直到按回车键为止。输入的内容存于＄500 开始的内存单元中。

2．A682

功能：读 1＃操纵器按键，键码存于＄90 单元。

3．A685

功能：读 0＃操纵器按键，键码存于＄90 单元。

第三节　电子钟程序设计

PC 机中有一个时钟程序，用户可以即时校对当前时间，可以随意控制时钟的显示与关闭，使用非常方便。

电脑学习机没有配备时钟软件，显得档次较低。能否为电脑学习机编写一个像 PC 中哪样的时钟程序呢？

以往虽然有介绍模拟电子钟的程序，但由于是用 F BASIC 语言编写的，运行时与其它 F BASIC 程序相冲突，不能并行工作(电子钟工作时则不能运行其它程序)。另外电子钟的时间取样是通过延时语句获得的，走时极不精确。所以，这类电子钟的实用价值不大。

在掌握了以上所学的内容后，我们可以着手编写真正的电子钟程序了。

一、编程思路

1. 以中断周期为时间基准(由于一个中断周期为 0.02 秒,因此只要设置一个中断计数器,就可以得到精确的时基)。

2. 用汇编语言编程,程序放在中断中处理,这样可以避免与用户程序的冲突,节省用户的程序存储空间。

3. 为了避免过多的现场保护,我们选取 $7FF0～$7FF3 作为计数、计秒、计分、计时器,以避开零页单元的使用。

4. 选定 $93 单元为电子钟的显示开关,$93=0 关显示;$93=1 开显示。

5. 为避免电子钟的时间显示破坏用户画面,把电子钟的时间显示座标选定在 F BASIC 显示画面之外。

二、编写汇编语言程序

根据以上编程思路,编写的汇编语言程序见 No.9－1～No.9－12。为使电子钟程序不占用机器语言程序仅有的 4K 资源($6000～$6FFF),我们把该程序的存放首址选为 $7E5F。

对于有些机型的系统可能使用该段地址空间,当电子钟运行时影响显示时,可把程序移到 $6E5F 开始的内存单元。改变程序地址时,注意把绝对跳转地址和数据暂存器作相应的改动。

No.9－1 画面定位子程序

```
7E5F   A5   9D        LDA   $9D
7E61   8D   05   20   STA   $2005
7E64   A5   69        LDA   $69
```

```
7E66  8D  05  20    STA  $2005
7E69  60            RTS
```

No.9－2 开显示子程序

```
7E6A  A5  33        LDA  $33
7E6C  8D  01  20    STA  $2001
7E6F  60            RTS
```

No.9－3 置 PPU 地址子程序

```
7E70  8D  06  20    STA  $2006
7E73  8E  06  20    STX  $2006
7E76  60            RTS
```

No.9－4 关显示子程序

```
7E77  20  80  7E    JSR  $7E80
7E7A  A9  00        LDA  $00
7E7C  8D  01  20    STA  $2001
7D7F  60            RTS
```

No.9－5 读中断子程序

```
7E80  AD  02  20    LDA  $2002
7E83  30  FB        BMI  $7E80
7E85  AD  02  20    LDA  $2002
7E88  10  FB        BPL  $7E85
7E8A  60            RTS
```

No.9－6 加法子程序

```
7E8B      18        CLC;
7E8C  A5  2C        LDA  $2C
7E8E  65  28        ADC  $28
7E90  85  28        STA  $28
7E92  A5  2D        LDA  $2D
7E94  65  29        ADC  $29
7E96  85  29        STA  $29;
```

$28、$29 中的内容加上 $2C、$2D 单元中的内容，结果仍存在 $28、$29 单元。

```
7E98  60              RTS
```

No. 9－7 减法子程序

```
7E99  38              SEC;        ┐
7E9A  A5  28          LDA  $28    | $28、$29 中的内
7E9C  E5  2C          SBC  $2C    | 容减去 $2C、$2D
7E9E  85  28          STA  $28    | 单元中的内容，结
7EA0  A5  29          LDA  $29    | 果仍存在 $28、$29
7EA2  E5  2D          SBC  $2D    | 单元。
7EA4  85  29          STA  $29;   ┘
7EA6  60              RTS
```

No. 9－8 16 进制数据转换成可显示的 ASCII 码子程序

```
7EA7  A9  00          LDA  # $00
7EA9  85  46          STA  $46
7EAB  A5  46          LDA  $46
7EAD  48              PHA
7EAE  A9  10          LDA  # $10   ;转换万位送入 $3D
7EB0  85  2C          STA  $2C
7EB2  A9  27          LDA  $27
7EB4  85  2D          STA  $2D
7EB6  20  F3  7E      JSR  $7EF3
7EB9  A9  E8          LDA  # $E8   ;转换千位送入 $3E
7EBB  85  2C          STA  $2C
7EBD  A9  03          LDA  # $03
7EBF  85  2D          STA  $2D
7EC1  20  F3  7E      JSR  $7EF3
7EC4  A9  64          LDA  # $64   ;转换百位送入 $3F
7EC6  85  2C          STA  $2C
7EC8  A9  00          LDA  # $00
7ECA  85  2D          STA  $2D
7ECC  20  F3  7E      JSR  $7EF3
```

```
7ECF  A9  0A       LDA  #$0A   ;转换十位送入$40
7ED1  85  2C       STA  $2C
7ED3  20  F3  7E   JSR  $7EF3
7ED6  A5  28       LDA  $28    ;转换个位送入$41
7ED8  18           CLC
7ED9  69  30       ADC  #$30
7EDB  20  EC  7E   JSR  $7EEC
7EDE  A9  00       LDA  #$00
7EE0  20  EC  7E   JSR  $7EEC
7EE3  68           PLA
7EE4  85  46       STA  $46
7EE6  60           RTS
```

No. 9—9 把$28、$29 单元的数据转换为十进制 ASCII 码子程序

```
7EE7  20  8B  7E   JSR  $7E8B   ;两字节加法运算
7EEA  A5  47       LDA  $47
7EEC  A6  46       LDX  $46;   ┐
7EEE  95  3D       STA  $3D,X  ├ 把 A 中的数据置入 $3D～$41 单元
7EF0  E6  46       INC  $46;   ┘
7FF2  60           RTS         ;一个 ASCII 码转换完毕,返回
7FF3  A9  30       LDA  #$30
7FF5  85  47       STA  $47    ;预置$47=30
7FF7  20  99  7E   JSR  $7E99  ;两字节减法运算
7FFA  90  EB       BCC  $7EE7  ;不够减转移
7FFC  E6  47       INC  $47    ;$47 存 ASCII 码
7FFE  4C  F7  7E   JMP  $7EF7  ;继续累减
```

No. 9—10 电子钟中断入口设置子程序

```
7F01  A9  0A       LDA  #$0A
```

```
7F03  85  EE        STA  $EE
7F05  A9  7F        LDA  #$7F
7F07  85  EF        STA  $EF   ;置中断程序入口$7F0A
7F09  60            RTS
```

No. 9－11 电子钟工作主控程序

```
7F0A  48            PHA
7F0B  20  12  7F    JSR  $7F12;调电子钟工作子程序
7F0E  68            PLA
7F0F  4C  0F  88    JMP  $880F
```

No. 9－12 电子钟工作子程序

```
7F12  EE  F0  7F    INC  $7FF0  ;计数
7F14  AD  F0  7F    LDA  $7FF0
7F16  C9  32        CMP  #$32
7F18  F0  01        BEQ  $7F1D  ;计数够1秒时转移
7F1A  60            RTS         ;不够1秒直接返回
7F1D  A9  00        LDA  #$00
7F1F  8D  F0  7F    STA  $7FF0  ;计数器置0
7F22  EE  F1  7F    INC  $7FF1  ;计秒器加1
7F25  AD  F1  7F    LDA  $7FF1
7F28  C9  3C        CMP  #$3C
7F2A  D0  23        BNE  $7F4F  ;不够1分转显示处理
7F2C  A9  00        LDA  #$00
7F2E  8D  F1  7F    STA  $7FF1  ;够1分,计秒器清0
7F31  EE  F2  7F    INC  $7FF2  ;计分器加1
7F34  AD  F2  7F    LDA  $7FF2
7F37  C9  3C        CMP  #$3C
7F39  D0  14        BNE  $7F4F  ;不够1小时转显示处理
7F3B  A9  00        LDA  #$00
```

```
7F3D  8D  F2  7F    STA   $7FF2   ;够1小时,计分器清0
7F40  EE  F3  7F    INC   $7FF3   ;计时器加1
7F43  AD  F3  7F    LDA   $7FF3
7F46  C9  0D        CMP   #$0D
7F48  90  05        BCC   $7F4F   ;小于12小时,转显示
7F4A  A9  00        LDA   #$00
7F4C  8D  F3  7F    STA   $7FF3   ;等于12小时,清计时器
7F4F  98            TYA;          ┐
7F50  48            PHA           │
7F51  8A            TXA           │
7F52  48            PHA           │ 保护现场
7F53  A2  1F        LDX   #$1F    │
7F55  B5  28        LDA   $28,X   │
7F57  48            PHA           ┘
7F58  CA            DEX           ┐
7F59  10  FA        BPL   $7F55;  ┘
7F5B  A0  00        LDY   #$00
7F5D  A9  02        LDA   #$02
7F5F  8D  F4  7F    STA   $7FF4   ;预置时间存储器偏址指针
7F62  AE  F4  7F    LDX   $7FF4   ;取偏址→X
7F65  BD  F1  7F    LDA   $7FF1,X;根据X依次取时、分、秒
7F68  85  28        STA   $28
7F6A  A9  00        LDA   #$00
7F6C  85  29        STA   $29     ;取一个时间值置入$28、$29
7F6E  20  A7  7E    JSR   $7EA7   ;转换16进制数为两位可显
                                  示的ASCII码存入$40、
                                  $41单元,$40存高位、
                                  $41为低位
```

```
7F71  A5 40     LDA  $40   ;把时间数据依次存入$7FF5
7F73  99 F5 7F  STA  $7FF5,Y   ;～$7FFD
7F76  C8        INY
7F77  A5 41     LDA  $41
7F79  99 F5 7F  STA  $7FF5,Y
7F7C  C8        INY
7F7D  A9 3A     LDA  #$3A
7F7F  99 F5 7F  STA    $7FF5,Y   ;把":"存入$7FF5～
                                 $7FFD中
7F82  C8        INY
7F83  CE F4 7F  DEC  $7FF4
7F86  AE F4 7F  LDX  $7FF4
7F89  10 DA     BPL  $7F65   ;时、分、秒未转换完,继续
7F8B  A5 93     LDA  $93   ;检查时间显示开关
7F8D  F0 1D     BEQ  $7FAC   ;关时间显示时,转移
7F8F  20 77 7E  JSR  $7E77   ;关显示
7F92  A0 00     LDY  #$00
7F94  A9 20     LDA  #$20
7F96  A2 36     LDX  #$36
7F98  20 70 7E  JSR  $7E70   ;置PPU地址为S2036
7F9B  B9 F5 7F  LDA  $7FF5,Y   ;┐
7F9E  8D 07 20  STA  $2007     ;│
7FA1  C8        INY            ;├送显时间
7FA2  C0 08     CPY  #$08      ;│
7FA4  D0 F5     BNE  $7F9B   ;┘
7FA6  20 6A 7E  JSR  $7E6A   ;开显示
7FA9  20 5F 7E  JSR  $7E5F   ;画面定位
```

```
7FAC  A2  00      LDX  #$00;
7FAE  68          PLA
7FAF  95  28      STA  $28,X
7FB1  E8          INX
7FB2  E0  20      CPX  #$20
7FB4  D0  F8      BNE  $7FAE
7FB6  68          PLA
7FB7  AA          TAX
7FB8  68          PLA
7FB9  A8          TAY  ;
7FBA  60          RTS
```

恢复现场

为压缩篇幅，请读者根据各指令的注释自己分析编程原理。

三、电子钟机器语言程序的 F BASIC 写入程序

若读者没有装入小监控程序，可编写一个 F BASIC 写入程序，把以上机器语言程序写入内存。程序见 No. 9－13。

```
5 REM No. 9－13 (电子钟写入程序)
10 FOR I=0 TO 347 : READ A$
20 POKE &7E5F + I,VAL("&" + A$)
30 NEXT
40 DATA A5,9D,8D,05,20,A5,69,8D,05,20,60,
   A5,33,8D,01,20,60,8D,06,20,8E,06,20,
   60,20,80,7E,A9,00,8D,01,20,60,AD,02,
   20,30,FB,AD, 02,20,10,FB,60,18,A5,2C,
   65,28,85,28,A5,2D,65,29,85,29,60
50 DATA 38,A5,28,E5,2C,85,28,A5,29,E5,2D,
   85,29,60,A9,00,85,46,A5,46,48,A9,10,
   85,2C,A9,27,85,2D,20,F3,7E,A9,E8,85,
```

```
2C,A9,03,85,2D,20,F3,7E,A9,64,85,2C,
A9,00,85,2D,20,F3,7E,A9,0A,85,2C,20,
F3,7E
60 DATA A5,28,18,69,30,20,EC,7E,A9,00,20,
EC,7E,68,85,46,60,20,8B,7E,A5,47,A6,
46,95,3D,E6,46,60,A9,30,85,47, 20,99,
7E,90,EB,E6,47,4C,F7,7E
70 DATA A9,0A,85,EE,A9,7F,85,EF,60,48,20,
12,7F,68,4C,0F,88,EE,F0,7F,AD,F0,7F,
C9,32,F0,01,60,A9,00,8D,F0,7F,EE,F1,
7F,AD,F1,7F,C9,3C,D0,23,A9,00,8D,F1,
7F,EE,F2,7F,AD,F2,7F,C9,3C,D0,14,A9,
00,8D,F2,7F
80 DATA EE,F3,7F,AD,F3,7F,C9,0D,90,05,A9,
00,8D,F3,7F,98,48,8A,48,A2,1F,B5,28,
48,CA,10,FA,A0,00,A9,02,8D,F4,7F,AE,
F4,7F,BD,F1,7F,85,28,A9,00,85,29,20,
A7,7E,A5,40,99,F5,7F,C8,A5,41,99,F5,
7F,C8,A9,3A,99,F5,7F,C8,CE,F4,7F,AE,
F4,7F,10,DA
90 DATA A5,93,F0,1D,20,77,7E,A0,00,A9,20,
A2,36,20,70,7E,B9,F5,7F,8D,07,20,C8,
C0,08,D0,F5,20,6A,7E,20,5F,7E,A2,00,
68,95,28,E8,E0,20,D0,F8,68,AA,68,A8,60
```

四、电子钟的运行

程序 No. 9－13 输入完毕，检查无误后，运行一次可以删除，以输入其它程序。也可以暂不删除，以便有错时修改。

电子钟的 F BASIC 运行程序见 No. 9－14。

```
5 REM No. 9－14（电子钟运行程序）
100 CALL 32513:POKE 147,1:CLS（启动电子钟）
110 INPUT "INPUT <hour> ";A:POKE 32755,A（调时）
120 INPUT "INPUT <minute>";A:POKE 32754,A(调分）
130 INPUT "INPUT <second>";A POKE 32753,A（调秒）
140 CLS:LOCATE 5,5:PRINT "WELWCOME USING";:LOCATE
    4 , 7:PRINT"ELECTRONIC CLOCK"
150 PAUSE 300:CLS:NEW
```

运行程序 No. 9－13 后，根据屏幕提示，依次输入当前的时间 时、分、秒，电子钟即进入计时运行。运行中欲停止电子钟的显示可执行“POKE 147，0”；若开显示可执行“POKE 147，1”；当欲改变计时的初始时间时可执行“POKE 32755，XX”以改变计时、“POKE 32754，XX”以改变计分、“POKE 32753，XX ”以改变计秒。从而可以使用它检测程序的运行时间等，使用十分方便。

五、电子钟的特点

1. 走时精度高，可与电子石英钟相媲美。

2. 与 F BASIC 程序并行工作，无论进行 F BASIC 编程或运行程序，都不影响电子钟的走时精度和显示。

3. 可随时进行时间校对，既可作电子钟使用又可用于电脑工作计时，还可用于程序运行时间的检测。

4. 电子钟的时间显示在屏幕的右上角，在 F BASIC 显示画面之外，不破坏 F BASIC 的显示画面。

第四节　电子钟程序的优化

第三节的电子钟程序完全独立，程序量较大，共占用了

348 个字节。如果在该程序中调用背景开、关显示、置屏幕、置 PPU 地址、十六进制数据转换为十进制数据等子程序，可使程序得到极大的简化。

一、机器语言优化程序清单

优化后的电子钟程序清单见 No. 9－15～No. 9－17。

No. 9－15 置中断入口子程序

```
7F01 A9 0A      LDA  # $0A
7F03 85 EE      STA  $EE
7F05 A9 7F      LDA  # $7F
7F07 85 EF      STA  $EF
7F09 60         RTS
```

No. 9－16 电子钟控制子程序

```
7F0A 48         PHA
7F0B 20 12 7F   JSR  $7F12
7F0E 68         PLA
7F0F 4C 0F 88   JMP  $880F
```

No. 9－17 电子钟工作子程序

```
7F12 EE F0 7F  INC  $7FF0 ;计数
7F14 AD F0 7F  LDA  $7FF0
7F16 C9 32     CMP  # $32
7F18 F0 01     BEQ  $7F1D
7F1A 60        RTS
7F1D A9 00     LDA  # $00
7F1F 8D F0 7F  STA  $7FF0
```

```
7F22 EE F1 7F  INC  $7FF1 ;计秒
7F25 AD F1 7F  LDA  $7FF1
7F28 C9 3C     CMP  #$3C
7F2A D0 23     BNE  $7F4F
7F2C A9 00     LDA  #$00
7F2E 8D F1 7F  STA  $7FF1
7F31 EE F2 7F  INC  $7FF2 ;计分
7F34 AD F2 7F  LDA  $7FF2
7F37 C9 3C     CMP  #$3C
7F39 D0 14     BNE  $7F4F
7F3B A9 00     LDA  #$00
7F3D 8D F2 7F  STA  $7FF2
7F40 EE F3 7F  INC  $7FF3 ;计时
7F43 AD F3 7F  LDA  $7FF3
7F46 C9 0D     CMP  #$0D
7F48 90 05     BCC  $7F4F
7F4A A9 00     LDA  #$00
7F4C 8D F3 7F  STA  $7FF3
7F4F 98        TYA ;          ┐
7F50 48        PHA            │
7F51 8A        TXA            │
7F52 48        PHA            │
7F53 A2 1F     LDX  #$1F      │ 保护现场
7F55 B5 28     LDA  $28,X     │
7F57 48        PHA            │
7F58 CA        DEX            │
7F59 10 FA     BPL  $7F55 ;   ┘
```

```
7F5B A0 00      LDY # $ 00 ;
7F5D A9 02      LDA # $ 02
7F5F 8D F4 7F   STA $ 7FF4
7F62 AE F4 7F   LDX $ 7FF4
7F65 BD F1 7F   LDA $ 7FF1,X
7F68 85 28      STA $ 28
7F6A A9 00      LDA # $ 00
7F6C 85 29      STA $ 29
7F6E 20 3C 8D   JSR $ 8D3C      存入 $ 3D～ $ 41
7F71 A5 40      LDA $ 40        先把十进制数转换为
7F73 99 F5 7F   STA $ 7FF5,Y    可显示的 ASCII 码
7F76 C8         INY             再把时间数据存入
7F77 A5 41      LDA $ 41        $ 7FF5～ $ 7FFD
7F79 99 F5 7F   STA $ 7FF5,Y
7F7C C8         INY
7F7D A9 3A      LDA # $ 3A
7F7F 99 F5 7F   STA $ 7FF5,Y
7F82 C8         INY
7F83 CE F4 7F   DEC $ 7FF4
7F86 AE F4 7F   LDX $ 7FF4
7F89 10 DA      BPL $ 7F65 ;
7F8B A5 93      LDA $ 93
7F8D F0 1D      BEQ $ 7FAC
7F8F 20 F1 AB   JSR $ ABF1 ;关显示
7F92 A0 00      LDY # S 00
7F94 A9 20      LDA # S 20
7F96 A2 36      LDX # $ 36
7F98 20 F8 81   JSR $ 81F8 ;置 PPU 地址 $ 2036
```

```
7F9B B9 F5 7F  LDA  $7FF5,Y ;┐
7F9E 8D 07 20  STA  $2007     │
7FA1 C8        INY            │ 送显时间
7FA2 C0 08     CPY  #$08      │
7FA4 D0 F5     BNE  $7F9B ;   ┘
7FA6 20 FD AB JSR  $ABFD ;开显示
7FA9 20 E6 AB JSR  $ABE6 ;画面定位
7FAC A2 00     LDX  #$00 ;┐
7FAE 68        PLA        │
7FAF 95 28     STA  $28,X │
7FB1 E8        INX        │
7FB2 E0 20     CPX  #$20  │ 恢复现场
7FB4 D0 F8     BNE  $7FAE │
7FB6 68        PLA        │
7FB7 AA        TAX        │
7FB8 68        PLA        │
7FB9 A8        TAY ;      ┘
7FBA 60        RTS
```

优化后的程序比原程序少占用 162 个单元，程序量压缩了 46·6%。可见，灵活的调用系统子程序可最大限度地优化程序。

二、优化电子钟的写入运行程序清单

优化后的电子钟 F BASIC 写入、运行程序清单见 No. 9－18。

```
5 REM No. 9－18（电子钟优化写入、运行程序）
10 FOR I=0 TO 185 :READ A$
20 POKE &7F01 + I,VAL( "&" + A$ )
30 NEXT
```

```
40 DATA A9,0A,85,EE,A9,7F,85,EF,60,48,20,12,7F,68,4C,0F,
   88,EE,F0,7F,AD,F0,7F,C9,32,F0,01,60,A9,00,8D,F0,7F,
   EE,F1,7F,AD,F1,7F,C9,3C,D0,23,A9,00,8D,F1,7F,EE,F2,
   7F,AD,F2,7F,C9,3C,D0,14,A9,00,8D,F2,7F
50 DATA EE,F3,7F,AD,F3,7F,C9,0D,90,05,A9,00,8D,F3,7F,
   98,48,8A,48,A2,1F,B5,28,48,CA,10,FA,A0,00,A9,02,8D,
   F4,7F,AE,F4,7F,BD,F1,7F,85,28,A9,00,85,29,20,3C,8D,
   A5,40,99,F5,7F,C8,A5,41,99,F5,7F,C8,A9,3A,99,F5,7F,
   C8,CE,F4,7F,AE,F4,7F,10,DA
60 DATA A5,93,F0,1D,20,F1,AB,A0,00,A9,20,A2,36,20,F8,81,
   B9,F5,7F,8D,07,20,C8,C0,08,D0,F5,20,FD,AB,20,E6,AB,
   A2,00,68,95,28,E8,E0,20,D0,F8,68,AA,68,A8,60
70 CALL 32513:POKE 147,1:CLS (启动电子钟)
80 INPUT "INPUT <hour> "; A :POKE 32755,A (调时)
90 INPUT "INPUT <minute>";A :POKE 32754,A (调分)
100 INPUT "INPUT <second>";A :POKE 32753,A (调秒)
110 CLS :LOCATE 5,5 :PRINT "WELCOME USING";:
    LOCATE 4, 7: PRINT "ELECTRONIC CLOCK"
120 PAUSE 300:CLS :NEW
```

关于6527汇编语言程序设计就介绍到这里。

通过以上九章的讨论，我们已经明确了6527汇编语言的结构、特点、指令系统组成和基本程序设计方法，并拥有了编程工具软件和较多的工具子程序。已能设计许许多多应用软件了。有关任天堂游戏中背景画面绘制、音响控制等程序的设计方法及更具体的应用，将在丛书之七、之八中继续讨论。

现在，建筑高楼大厦的砖瓦石料就摆在你的面前，至于建造什么样的琼楼玉宇、舞榭歌台，就看你如何安排堆砌了。

附 录 6527指令系统表

符号 指令	操作 过程	机器码 (16进制)	N	T	标志寄存器P NV..DIZC	寻址 方式
ADC # $n	A+$n+C→A	69 n	2	2	NV....ZC	立即寻址
ADC $n2n1	A+($n2n1)+C→A	6D n1 n2	3	4	NV....ZC	绝对寻址
ADC $n	A+($n)+C→A	65 n	2	3	NV....ZC	零页寻址
ADC ($n,X)	A+(ADDRES)+C→A	61 n	2	6	NV....ZC	先变址间接
ADC ($n),Y	A+(ADDRES)+C→A	71 n	2	5	NV....ZC	后变址间接
ADC $n,X	A+($n+X)+C→A	75 n	2	4	NV....ZC	零页X变址
ADC $n2n1,X	A+($n2n1+X)+C→A	7D n1 n2	3	4a	NV....ZC	绝对X变址
ADC $n2n1,Y	A+($n2n1+Y)+C→A	79 n1 n2	3	4a	NV....ZC	绝对Y变址
AND # $n	A∧$n→A	29 n	2	2	N.....Z.	立即寻址
AND $n2n1	A∧($n2n1)→A	2D n1 n2	3	4a	N.....Z.	绝对寻址
AND $n	A∧($n)→A	25 n	2	3	N.....Z.	零页寻址
AND ($n,X)	A∧(ADDRES)→A	21 n	2	6	N.....Z.	先变址间接
AND ($n),Y	A∧(ADDRES)→A	31 n	2	5	N.....Z.	后变址间接
AND $n,X	A∧($n+X)→A	35 n	2	4	N.....Z.	零页X变址
AND $n2n1,X	A∧($n2n1+X)→A	3D n1 n2	3	4a	N.....Z.	绝对X变址
AND $n2n1,Y	A∧($n2n1+Y)→A	39 n1 n2	3	4a	N.....Z.	绝对Y变址
ASL $n2n1		0E n1 n2	3	6	N.....ZC	绝对寻址
ASL $n	C→D7→D0→0	06 n	2	5	N.....ZC	零页寻址
ASL	M	0A	1	2	N.....ZC	累加器寻址
ASL $n,X		16 n	2	6	N.....ZC	零页X变址
ASL $n2n1,X		1E n1 n2	3	7	N.....ZC	绝对X变址
BCC $n	C=0分支	90 n	2	3b	.....	相对寻址
BCS $n	C=1分支	B0 n	2	3b	.....	相对寻址
BEQ $n	Z=1分支	F0 n	2	3b	.....	相对寻址
BNE $n	Z=0分支	D0 n	2	3b	.....	相对寻址
BVC $n	V=0分支	50 n	2	3b	.....	相对寻址
BVS $n	V=1分支	70 n	2	3b	.....	相对寻址

续表(一)

符号 指令	操作 过程	机器码 (16进制)	N	T	标志寄存器 P NV..DIZC	寻址 方式
BMI $n	N=1 分支	30 n	2	3b	.....	相对寻址
BPL $n	N=0 分支	10 n	2	3b	.....	相对寻址
BIT $n	A∧($n)	24 n	2	3	NV....Z.	零页寻址
BIT $n2n1	A∧($n2n1)	2C n1 n2	3	4	NV....Z.	绝对寻址
BRK	中断	00	1	7	...1.1..	隐含寻址
CLC	0→C	18	1	2	0	隐含寻址
CLD	0→D	D8	1	2	....0...	隐含寻址
CLI	0→I	58	1	2	.....0..	隐含寻址
CLV	0→V	B8	1	2	.0.....	隐含寻址
CMP #$n	A − $n	C9 n	2	2	N.....ZC	立即寻址
CMP $n2n1	A − ($n2n1)	CD n1 n2	3	4	N.....ZC	绝对寻址
CMP ($n,X)	A − (ADDRES)	C1 n	2	6	N.....ZC	先变址间接
CMP ($n),Y	A − (ADDRES)	D1 n	2	5	N.....ZC	后变址间接
CMP $n,X	A − ($n+X)	D5 n	2	4	N.....ZC	零页X变址
CMP $n2n1,X	A − ($n2n1+X)	DD n1 n2	3	4a	N.....ZC	绝对X变址
CMP $n2n1,Y	A − ($n2n1+Y)	D9 n1 n2	3	4a	N.....ZC	绝对Y变址
CMP $n	A − ($n)	C5 n	2	3	N.....ZC	零页寻址
CPX #$n	X − $n	E0 n	2	2	N.....ZC	立即寻址
CPX $n	X − ($n)	E4 n	2	3	N.....ZC	零页寻址
CPX $n2n1	X − ($n2n1)	EC n1 n2	3	4	N.....ZC	绝对寻址
CPY #$n	Y − $n	C0 n	2	2	N.....ZC	立即寻址
CPY $n	Y − ($n)	C4 n	2	3	N.....ZC	零页寻址
CPY $n2n1	Y − ($n2n1)	CC n1 n2	3	4	N.....ZC	绝对寻址
DEC $n	($n)−1→($n)	C6 n	2	5	N.....Z.	零页寻址
DEC $n,X	($n+X)−1→($n+X)	D6 n	2	6	N.....Z.	零页X变址
DEC $n2n1	($n2n1)−1→($n2n1)	CE n1 n2	3	6	N.....Z.	绝对寻址
DEC $n2n1,X	($n2n1+X)−1→($n2n1+X)	DE n1 n2	3	7	N.....Z.	绝对X变址
DEX	X − 1→X	CA	1	2	N.....Z.	隐含寻址

续表(二)

符号 指令	操作 过程	机器码 (16 进制)	N	T	标志寄存器 P NV..DIZC	寻址 方式
DEY	Y − 1 →Y	88	1	2	N.....Z.	隐含寻址
EOR # $n	A⊙$n→ A	49 n	2	2	N.....Z.	立即寻址
EOR $n	A⊙($n)→ A	45 n	2	3	N.....Z.	零页寻址
EOR $n,X	A⊙($n+X) →A	55 n	2	4	N.....Z.	零页 X 变址
EOR $n2n1	A⊙($n2n1) →A	4D n1 n2	3	4	N.....Z.	绝对寻址
EOR $n2n1,X	A⊙($n2n1+X) →A	5D n1 n2	3	4a	N.....Z.	绝对 X 变址
EOR $n2n1,Y	A⊙($n2n1+Y) →A	59 n1 n2	3	4a	N.....Z.	绝对 Y 变址
EOR ($n,X)	A⊙(ADDRES) →A	41 n	2	6	N.....Z.	先变址间接
EOR ($n),Y	A⊙(ADDRES) →A	51 n	2	5c	N.....Z.	后变址间接
INC $n	($n)+1→ ($n)	E6 n	2	5	N.....Z.	零页寻址
INC $n,X	($n+X)+1→($n+X)	F6 n	2	6	N.....Z.	零页 X 变址
INC $n2n1	($n2n1)+1 → ($n2n1)	EE n1 n2	3	6	N.....Z.	绝对寻址
INC $n2n1,X	($n2n1+X)+1 → ($n2n1+X)	FE n1 n2	3	7	N.....Z.	绝对 X 变址
INX	X + 1 → X	E8	1	2	N.....Z.	隐含寻址
INY	Y + 1 → Y	C8	1	2	N.....Z.	隐含寻址
JMP $n2n1	$n2n1 →PC	4C n1 n2	3	3	.....	绝对寻址
JMP ($n2n1)	($n2n1+1, $n2n1) → PC	6C n1 n2	3	5	.....	间接寻址
JSR $n2n1	$n2n1 → PC	20 n1 n2	3	6	.....	绝对寻址
LDA # $n	$n →A	A9 n	2	2	N.....Z.	立即寻址
LDA $n	($n) →A	A5 n	2	3	N.....Z.	零页寻址
LDA $n,X	($n+X) →A	B5 n	2	4	N.....Z.	零页 X 变址
LDA $n2n1	($n2n1) →A	AD n1 n2	3	4	N.....Z.	绝对寻址
LDA $n2n1,X	($n2n1+X) →A	BD n1 n2	3	4a	N.....Z.	绝对 X 变址
LDA $n2n1,Y	($n2n1+Y) →A	B9 n1 n2	3	4a	N.....Z.	绝对 Y 变址
LDA ($n,X)	(ADDRES) →A	A1 n	2	6	N.....Z.	先变址间接
LDA ($n),Y	(ADDRES) →A	B1 n	2	5c	N.....Z.	后变址间接

续表(三)

符号 指令	操作 过程	机器码 (16进制)	N	T	标志寄存器 P NV..DIZC	寻址 方式
LDX # $n	$n →X	A2 n	2	2	N.....Z.	立即寻址
LDX $n	($n) →X	A6 n	2	3	N.....Z.	零页寻址
LDX $n,Y	($n+Y) →X	B6 n	2	4	N.....Z.	零页Y变址
LDX $n2n1	($n2n1) →X	AE n1 n2	3	4	N.....Z.	绝对寻址
LDX $n2n1,Y	($n2n1+Y) →X	BE n1 n2	3	4a	N.....Z.	绝对Y变址
LDY # $n	$n →Y	A0 n	2	2	N.....Z.	立即寻址
LDY $n	($n) →Y	A4 n	2	3	N.....Z.	零页寻址
LDY $n,X	($n+X) →Y	B4 n	2	4	N.....Z.	零页X变址
LDY $n2n1	($n2n1) →Y	AC n1 n2	3	4	N.....Z.	绝对寻址
LDY $n2n1,X	($n2n1+X) →Y	BC n1 n2	3	4a	N.....Z.	绝对X变址
LSR		4A	1	2	0.....ZC	累加器寻址
LSR $n		46 n	2	5	0.....ZC	零页寻址
LSR $n,X	0 →D7 →D0 →C	56 n	2	6	0.....ZC	零页X变址
LSR $n2n1	M	4E n1 n2	3	6	0.....ZC	绝对寻址
LSR $n2n1,X		5E n1 n2	3	7	0.....ZC	绝对X变址
NOP		EA	1	2	.....	隐含寻址
OR # $n	A ∨ $n →A	09 n	2	2	N.....Z.	立即寻址
ORA $n	A ∨ ($n) →A	05 n	2	3	N.....Z.	零页寻址
ORA $n,X	A ∨ ($n+X) →A	15 n	2	4	N.....Z.	零页X变址
ORA $n2n1	A ∨ ($n2n1) →A	0D n1 n2	3	4	N.....Z.	绝对寻址
ORA $n2n1,X	A ∨ ($n2n1+X) →A	1D n1 n2	3	4a	N.....Z.	绝对X变址
ORA $n2n1,Y	A ∨ ($n2n1+Y) →A	19 n1 n2	3	4a	N.....Z.	绝对Y变址
ORA ($n,X)	A ∨ (ADDRES) →A	01 n	2	6	N.....Z.	先变址间接
ORA ($n),Y	A ∨ (ADDRES) →A	11 n	2	5	N.....Z.	后变址间接
PHA	A →Ms,S−1 →S	48	1	3	.....	隐含寻址
PHP	P →Ms,S−1 →S	08	1	3	.....	隐含寻址
PLA	S+1 →S,Ms →A	68	1	4	.....	隐含寻址
PLP	S+1 →S,Ms →P	28	1	4	恢复P原来的状态	隐含寻址

续表(四)

符号 指令	操作 过程	机器码 (16进制)	N	T	标志寄存器 P NV..DIZC	寻址 方式
ROL		2A	1	2	N.....ZC	累加器寻址
ROL $n		26 n	2	5	N.....ZC	零页寻址
ROL $n,X	←D7←D0←C←	36 n	2	6	N.....ZC	零页X变址
ROL $n2n1	M	2E n1 n2	3	6	N.....ZC	绝对寻址
ROL $n2n1,X		3E n1 n2	3	7	N.....ZC	绝对X变址
ROR		6A	1	2	N.....ZC	累加器寻址
ROR $n		66 n	2	5	N.....ZC	零页寻址
ROR $n,X	→C→D7→D0→	76 n	2	6	N.....ZC	零页X变址
ROR $n2n1	M	6E n1 n2	3	6	N.....ZC	绝对寻址
ROR $n2n1,X		7E n1 n2	3	7	N.....ZC	绝对X变址
RTI	中断返回	40	1	6	恢复P原来的状态	隐含寻址
RTS	子程序返回	60	1	6	.....	隐含寻址
SBC # $n	A←$n−$\overline{C}$→A	E9 n	2	2	NV.....C	立即寻址
SBC $n	A←($n)−$\overline{C}$→A	E5 n	2	3	NV.....C	零页寻址
SBC $n,X	A←($n+X)−$\overline{C}$→A	F5 n	2	4	NV.....C	零页X变址
SBC $n2n1	A←($n2n1)−$\overline{C}$→A	ED n1 n2	3	4	NV.....C	绝对寻址
SBC $n2n1,X	A←($n2n1+X)−$\overline{C}$→A	FD n1 n2	3	4a	NV....ZC	绝对X变址
SBC $n2n1,Y	A←($n2n1+Y)−$\overline{C}$→A	F9 n1 n2	3	4a	NV....ZC	绝对Y变址
SBC ($n,X)	A←(ADDRES)−$\overline{C}$→A	E1 n	2	6	NV....ZC	先变址间接
SBC ($n),Y	A←(ADDRES)−$\overline{C}$→A	F1 n	2	5c	NV....ZC	后变址间接
SEC	1→C	38	1	2	1	隐含寻址
SED	1→D	F8	1	2	....1...	隐含寻址
SEI	1→I	78	1	2	.....1..	隐含寻址
STA $n	A→($n)	85 n	2	3	.....	零页寻址
STA $n,X	A→($n+X)	95 n	2	4	.....	零页X变址
STA $n2n1	A→($n2n1)	8D n1 n2	3	4	.....	绝对寻址
STA $n2n1,X	A→($n2n1+X)	9D n1 n2	3	5	.....	绝对X变址
STA $n2n1,Y	A→($n2n1+Y)	99 n1 n2	3	5	.....	绝对Y变址
STA ($n,X)	A→(ADDRES)	81 n	2	6	.....	先变址间接
STA ($n),Y	A→(ADDRES)	91 n	2	6	.....	后变址间接

续表(五)

符号 指令	操作 过程	机器码 (16进制)	N	T	标志寄存器 P NV..DIZC	寻址 方式
STX $n	X →($n)	86 n	2	3	.....	零页寻址
STX $n,Y	X →($n+Y)	96 n	2	4	.....	零页Y变址
STX $n2n1	X →($n2n1)	8E n1 n2	3	4	.....	绝对寻址
STY $n	Y →($n)	84 n	2	3	.....	零页寻址
STY $n,X	Y →($n+X)	94 n	2	4	.....	零页X变址
STY $n2n1	Y →($n2n1)	8C n1 n2	3	4	.....	绝对寻址
TAX	A →X	AA	1	2	N.....Z.	隐含寻址
TAY	A →Y	A8	1	2	N.....Z.	隐含寻址
TSX	S →X	BA	1	2	N.....Z.	隐含寻址
TXA	X →A	8A	1	2	N.....Z.	隐含寻址
TXS	X →S	9A	1	2	.....	隐含寻址
TYA	Y →A	98	1	2	N.....Z.	隐含寻址

说明：1. 表中的符号

N——字节数。

T——时钟周期数。

n ——代表两位十六进制数。

n1n2——代表四位十六进制数。

ADDRES——代表有效地址。

⊙——表示异或运算

2. 表中的注释

a——如果跨页，指令执行时钟周期数为5。

b——如果跨页，指令执行时钟周期数为4；不分支时T=4。

c——如果跨页，指令执行时钟周期数为6。

电脑学习机上机指导系列丛书

1. 操作入门
2. 文字处理
3. 游戏BASIC语言
4. 中文BASIC语言
5. LOGO语言
6. 汇编语言
7. 电脑作曲
8. 电脑绘图
9. BASIC语言程序集
10. 硬件与维修

ISBN 7-5053-3530-8

9 787505 335301 >

责任编辑：施玉新

封面设计：阎欢玲

ISBN7-5053-3530-8/TP·1426 定价：12.00 元